CHALMERS

Real time systems

Collection of examples (2008-01-22)

In Ada:
A specification tells what you are intending to do. As a prerequisite you are to know what you are doing.
A declaration tells what you are actually doing.
Are you sure that what you do is what you really wanted to do?

Ada helps you a lot but can't do all of the work.
These assignments will hopefully get you on the track of really doing what you really wanted to do.

Contents

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg 2008

Parallel programming in Ada95

The following piece of code implements two simple tasks, *Producer* and *Consumer*, in Ada95. You can use this code together with your solutions from assignments 1-5. Create a source code file with the following client tasks. Then create working programs according to assignments 1-5 below.

```
--
-- Producer/Consumer client
--
-- uncomment clauses for the package you are working with
--with MAILBOX_HANDLER; use MAILBOX_HANDLER;    -- 1
--with NOTICE_HANDLER_1; use NOTICE_HANDLER_1;  -- 2
--with NOTICE_HANDLER_2; use NOTICE_HANDLER_2;  -- 3
--with NOTICE_HANDLER_3; use NOTICE_HANDLER_3;  -- 4

with TEXT_IO; use TEXT_IO;

procedure ProdConClient is
   task PRODUCER;
   task CONSUMER;


task body PRODUCER is
   c : MSG_TYPE := 'A';
begin
   loop
      delay( 4.0 );
      if c = 'Z' then
         c := 'A';
      end if;
         PUT_MESSAGE( c );
         c := MSG_TYPE'Succ(c);
   end loop;
end;

task body CONSUMER is
      c : MSG_TYPE := '?';

begin
   loop
      delay( 1.0 );
      --      delay( 5.0 );         -- use with notice_handler_3
      GET_MESSAGE( c );
      put( character( c ) );
   end loop;
end;

begin
   Put_Line( "Starting Producer/Consumer client...");
   null;
end;
```

Tasks and process communication

1. Write a task, which realizes a simple “mailbox”.
   The task has two entries, “put” and “get”. A message (“letter”) can be dropped into (“put”) or picked from (“get”) the mailbox by other tasks. The task should be designed so that each message that is delivered (by “put”) is consumed through a call to “get” before a new message is accepted. I.e. there is only room for one message at the same time and the message can only be read once.

2. Write a task, which realizes a simple notice-board.
   There is only room for a single message at the same time on this board. The task has two entries, PUT to place a message on the board and GET to read the current message.
   When a message is submitted to the board through PUT, any previously message is destroyed.
   Client tasks can read messages from a non-empty board. I.e. the same message may be read multiple times. If no message is available, then it cannot be read.
   PUT and GET should be accepted in arbitrary order but only one of them, at the same time.

3. Rewrite your solution from 2 so that entry calls to PUT is preferable to calls to GET

4. Rewrite your solution from 3 so that any message older than five seconds is thrown away assuming “age” is counted from last PUT or GET.

5. These assignments appeals to former assignment 2.
   A task is *repetitively* performing a certain amount of work through calling the procedure `WORK` (no parameters). `WORK` is supposed to be externally defined.
   - Each time `WORK` has been performed, this task should check the notice board for any pending messages.
   - If a new message is pending it should be passed to an externally defined procedure: `HANDLE_MESSAGE(msg).`
   - If no new message is pending this task should immediately continue its work.

   Write the repetition statements (loop) which accomplish this.

Task types

6. A real-time-system comprises at least *N* producer tasks that produces messages of a given type (`MESSAGE`) .
   - To each producer task there shall be an instance of a `MESSAGE_TASK`, which hosts a message box for one message, thus there might be *N* simultaneously pending messages.
   - There shall be one consumer task, which searches the *N* message boxes in order. When a message is available it shall be dispatched to the given procedure HANDLE_MESSAGE(MESSAGE). The search continues until all message boxes has been visited.
   - If all messages boxes are empty as the result of one search, the consumer process shall delay 20 seconds before the next search.

   Declare the tasks and write their bodies.

Protected Objects

7. Write a protected object, which realizes a simple notice-board.
There is only room for a single message at the same time on this board.
The object shall have two entries, `PUT_MESSAGE` to place a message on the board and `GET_MESSAGE` to read the current message.
When a message is submitted to the board through `PUT_MESSAGE`, any previously message is destroyed.
Client tasks can read messages from a non-empty board. I.e. the same message may be read multiple times. If no message is available, then it cannot be read. `PUT_MESSAGE` and `GET_MESSAGE` should be accepted in arbitrary order but only one of them, at the same time.

8. Three integer variables is shared by several tasks. Write an ADA package `Notice_Board` contain *read* and *write* operations by concurrent tasks, for these variables. The following type is declared:

```
type var_num is range 1 .. 3;
```

The package shall include the following procedures:

```
procedure read (num : in var_num; value : out integer)
     -- Returns value of variable denoted by 'var_num'.
     -- Block the calling task if the variable
     -- not have been previously assigned through 'write'.

procedure write (num : in var_num; value : in integer)
     -- Assign 'value' to variable denoted by 'var_num'.
```

Write operations are mutual exclusive for a particular variable, i.e. writing one variable should not block operations on another variable. Hint: Create a protected object for each variable).

9. A four road crossing is guarded by two pairs of traffic light. Only one pair is allowed to show green light at the same time. An ADA object `junction` is designed to permit a user task to turn green light on through a blocking primitive `Take`. Then eventually, when the task has switched the traffic light to red, this permission is returned through the primitive Release.
Implement the object `junction` within a procedure called `Traffic` as:
   a) a task.
   b) a protected object.

10. Write an ADA package `resource` for administration of ten identical resources. A client task should be able to aquire a range of sequential resources, e.g. range 1..5. Use protected objects. The following procedures shall be visible for client tasks.

```
type res_range is range 1..10;

procedure acquire(nr_of_resources : in res_range);
    -- Request 'nr_of_resources' resources.
    -- Return when the request can be granted.
    -- Block caller until request can be granted.

procedure trytoacquire(nr_of_resources : in res_range; ok :out
                       boolean);
    -- Request 'nr_of_resources' resources.
    -- Return 'ok' = true if request can be granted.
    -- Return 'ok' = false otherwise.
    -- This is a NON blocking primitive

procedure release(nr_of_resources : in res_range);
    -- Return 'nr_of_resources' resources assuming
    -- that they have been previusly granted
    -- through 'aquire'.
```

Monitors and semaphores

11. Monitors has several similarities with Ada95 ”protected objects”. Monitors are not supported by Ada95 however they are of general interest in parallel programming.
A monitor guarantees mutual exclusion. A monitor can export procedures and functions but not entries. Entries can be simulated using so called “condition variables”. There are three possible operations on condition variables.
    - Wait(Condition_variable) – calling task is queued (FIFO) on “Condition_variable”.
    - Signal(Condition_variable) – first task in corresponding queue becomes eligble for execution (is ”waked up”). A signal operation on an empty queue has no effect.
    - Non_Empty(Condition_variable) – this function returns Boolean TRUE if there is at least one task in the corresponding queue, returns FALSE otherwise.

    Suppose the above operations are provided by Ada and that `monitor` is present in the language.
    Implement a notice board using a `monitor` *`Board`*.
    There is only room for a single message at the same time on this board. The `monitor` exports two procedures, `Putmsg` to place a message on the board and `Getmsg` to read the current message.
    When a message is submitted to the board through `Putmsg`, any previously message is destroyed.
    Client tasks can read messages from a non-empty board. I.e. the same message may be read multiple times. If no message is available, then it cannot be read.
    `Putmsg` and `Getmsg` should be accepted in arbitrary order but only one of them, at the same time.

12. As with monitors, semaphores are of general interest in parallel programming. Semaphores, unlike monitors can be implemented regardless of programming language. Assume we have an Ada implementation of semaphores:

```
package semaphores is
   type Semaphore is ...—- useful type...
   procedure wait(Semaphore: in s);
   procedure signal(Semaphore: in s);
end semaphores;
```

    Implement a package `Board` using semaphores.
    There is only room for a single message at the same time on this board. The package exports two procedures, `Putmsg` to place a message on the board and `Getmsg` to read the current message.
    When a message is submitted to the board through `Putmsg`, any previously message is destroyed.
    Client tasks can read messages from a non-empty board. I.e. the same message may be read multiple times. If no message is available, then it cannot be read.
    `Putmsg` and `Getmsg` should be accepted in arbitrary order but only one of them, at the same time.

13. The control system in an automatic barbershop has tasks for a *cutting machine*, a *doorkeeper* and a *conveyor belt*. These functions shall be synchronised by use of the following procedures:

```
procedure Arrival;
procedure Removal;
procedure Start_Cut;
procedure Cutfinished;
```

    `Arrival` is called from the *doorkeeper* each time a new customer arrives.
    `Removal` is called from the *conveyor*, the call should be blocked if previous customer cutting is unfinished.
    `Start_cut` is called from the *cutting machine* prior to start work, `Cutfinished` is then called after servicing the customer. If there are no customer present when `Start_cut` is called, the task should be blocked.

    a) Implement the procedures with use of monitor constructions.
    b) Implement the procedures with semaphores.

14. A roller-coaster is monitored and controlled by a real-time system. The control program consists of several tasks. E.g. one task manages door opening/closing of vehicles, another task controls vehicle speed. Sensors and actuators are connected through a single interface, represented by (among others) the software routines:

```
      function is_closed return boolean;
      procedure start_vehicle;
      procedure close_doors;
```

For safety reasons, the roller-coaster should not start unless all doors are closed.
Implement procedure `Start` and `Close` in a package `SafeControl:`

a) Assuming monitors (as below) are available.

```
monitor m is
      procedure a;
      procedure b;
      c: Condition_Variable;
   end Acommands;

   monitor body m is

      procedure a is
      begin
            Wait(c);
      end a;

      procedure b is
      begin
            Send(c);
      end b;

   end m;
```

b) Assuming semaphores are available (example use below).

```
    S: semaphore;
    begin
      wait(S);
      ....
      signal(S);
    end a;
```

15. Consider a real-time system with seven tasks (P1-P7) and two semaphores (S1 and S2).
The tasks make use of semaphores due to the following:

P1	P2	P3	P4	P5	P6	P7

...
wait(S1)
	...
	signal(S2);
	...
	wait(S1);
	...
		...
		signal(S1);
		...
		wait(S2);
		...
		wait(S1);
			...
			wait(S1);
			...
				...
				signal(S2);
				...
				signal(S1);
				...
				wait(S1);
					...
					signal(S1);
					...
					wait(S2);
					...
						...
						signal(S2);
						...
						wait(S1);
						...

At a certain moment, just before the above scenario, the tasks are in the following states:

P1	P2	P3	P4	P5	P6	P7
RUNNING	RUNNABLE	RUNNABLE	RUNNABLE	WAITING ON S1	WAITING ON S1	WAITING ON S2

Show how the task's states are changed (RUNNING, RUNNABLE and WAITING ON Sx) from that moment and on.

Advanced assignments

16. *The dining philosophers*:
    a) Implement a solution to the dining philosophers problem where eating is blocked unless both eating sticks are available at the same time. I.e. it is not allowed to pick up a single stick and then put it back just because the other stick is unavailable. A single resource handler should be used to manage use of all sticks. Use Ada *requeue* in your solution.
    b) Why do we have to use *requeue* in this case?

17. The limit for a footbridge for tourists is 10 individuals. Each tourist request admittance to the footbridge and groups of 10 tourists are allowed to enter the bridge at one time. All tourists then leave the bridge at the same time.

    The typical behavior of a tourist when passing the footbridge is:

```
:
Take_Photograph;
Bridge.Request_W;
Walk;
Bridge.Release;
:
```

    Design a task, *bridge* that manages the tourist's tour on the bridge.
    Implement *bridge* as:
    a) A task.
    b) A protected object.

    -- flyttas, ej advanced

18. *Implement a protected object*. The object shall manage a *circular buffer* holding 8 items of type *Data*. The object shall have two entries:
    - Entry `PUT` is used to add items to the buffer. If there is unsufficient room, the caller shall be blocked until the buffer can accept the request.
    - Entry `GET` is used to pick items from the buffer. If the number of items in the buffer is to small to fulfil a request, the caller shall be blocked until the request can be granted.

19. *Implement a protected object*. A *supervisor robot* delievers parts to several *assembly robots*. The robots are controlled through an Ada task within each robot. A synchronising mechanism is now required, i.e. to prevent that an assembly robot tries to assemble details before they are available. We can accomplish this by providing a *mailbox*. The mailbox receives information of arrived parts (*detail_type* and *id*) as they are delievered from the supervising robot (`Put_Detail`). Before trying to assemble, the assembly robot shall check for the parts availability (`Get_Detail`), and if available, register the parts *id*. The mailbox managing of parts (details) shall be implemented in a stack-like fashion, i.e. last in – first out (LIFO).

    a) Suppose that only a single type detail exists.
    b) Suppose that arbitrary types of detail exist.
    c) Suppose that several supervisor robots exists, extend the solution but make sure that supervisor robots always is preffered to assembly robots.

    Use INTEGER as type for both *detail_type* and *id*.

    -- *Föreslår att följande uppgift utgår, eller ersättes med mer relevant exempel på pekar-hantering i Ada.*

20. En server-process skall efter att ha fått indata bearbeta detta och senare returnera utdata via ett anrop till en brevlåda liknande den i uppgift 1. Adressen till denna brevlåda skall ges i anropet av entryt:

```
entry Request(Box: Mailpek; Given_Info: Msg_Type);
```

    Bearbetningen av indata består i ett anrop av proceduren

```
process(info: Msg_Type);
```

    Visa hur Mailpek skall deklareras samt implementera serverprocessen.

Exception handling in Ada95

21. Show a procedure `main`, (with a `NULL` body) that takes care of the five common system exceptions:
    ```
    Constraint_Error
    Numeric_Error
    Program_Error
    Storage_Error
    Tasking_Error
    ```
    The exception handler should provide appropriate printouts to the system console, i.e. by printing out the cause of the exception.

22. Show a procedure `X` (with a NULL body) that takes care of an application defined exception `App_Exception` in a particular way (by printing out “Procedure X: App_exception”) and all other exceptions in a common way (by printing out “Procedure X: *exception name and exception message*”). It is assumed that the `App_Exception` only can be raised *within* the procedure.

23. Ada95 allows the application programmer to define any handling of exceptional events. Give an example of how you, as the programmer should handle the first instance of a particular exception, but would propagate a second occurrence of the same exception.

Low level programming in Ada95

Basic bit-operations and the unchecked conversions

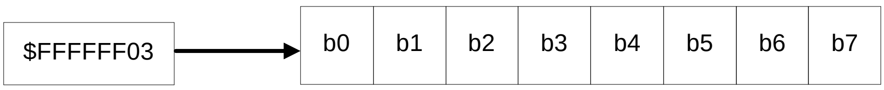
24. A “nibble” is 4 bits of a byte. I.e. a byte consists of a high nibble and a low nibble:

High nibble				Low nibble			
b7	b6	b5	b4	b3	b2	b1	b0

A byte has the same size as the data type `character` in Ada (8 bits). A character may represent the decimal values 0 through 255.

a) Show a declaration of a new type `BYTE`, with exactly the same properties as a character.
b) Using records show a type declaration `NIBBLE_TYPE`, which represents a byte with nibbles according to the figure. Recall that Ada syntax uses little endian bit numbering order.
c) Show a “nibble-swap” operation, i.e., an initial variable `a` of type `NIBBLE_TYPE` is assigned to a secondary variable `b` of the same type, but with another nibble order.

25. Assume an eight bit data register available at address FFFFFF03h in memory space (object `D_reg`). Show appropriate declarations and a function `ReadRegister`, which returns the value in this register.



26. Assume an 8-bit IO-port according to the following figure:

Rdy	Data						Ack
b7	b6	b5	b4	b3	b2	b1	b0

Bit ordering is “BIG ENDIAN”, i.e. reading left to right you will see the most significant bit first.
Rdy bit “set” (=1) reflects a ready status, i.e. there is valid data in bits 6 to 1. The ready bit can only be cleared by setting (writing ‘1’ to) bit *Ack*.

a) Show an appropriate type declaration `PORT` for this port using records. Recall that Ada syntax uses little endian bit numbering order.
b) Show an instance `port_inst` of this type at the physical address 0xFFFFFFF00.
c) Using a) and b), write a function `X` that returns TRUE if *Rdy* is set and FALSE otherwise.
d) Using a) and b), assuming that *Rdy* is set, write a function `Y` that returns data (as is) from the port.
e) Using a) and d), write a procedure `Z` that acknowledges a successful read, i.e. clears the *Ack* bit.
f) Conclude your experiences from these assignments. What did you find difficult?
What did you find trivial? According to your intuition, how would you separate your solutions into .ads files (ADa specification) and .adb (AD body) files?

27. Assume the following declarations:

```
-- nybble.ads
with System.Storage_elements;

package NYBBLE is
type BYTE is range 0..255; -- Named type and min..max values
type HIGH_NIBBLE_TYPE  is range 0..15; -- Named type and min..max values
type LOW_NIBBLE_TYPE is range 0..15; -- Named type and min..max values
type NIBBLES  is
      record
            high_nibble: HIGH_NIBBLE_TYPE;
            low_nibble : LOW_NIBBLE_TYPE;
      end record;

      for NIBBLES use
      record
            high_nibble at 0 range 0..3;  -- means b7-b4 in big endian
            low_nibble  at 0 range 4..7;  -- means b3-b0 in big endian
      end record;

      D_reg: BYTE;

      for D_reg'address use constant System.address :=
      System.Storage_elements.to_address(16#FFFFFF15#);

      procedure wnibble ( W : HIGH_NIBBLE_TYPE );
      procedure wnibble ( W : LOW_NIBBLE_TYPE );

end NYBBLE;
```

We now want a "single" procedure `wnibble(..)` to write either the value of *high nybble* or the value of *low nybble* of a byte to the register located at FFFFFF15. Show how to do this using function overloading and unchecked conversions.

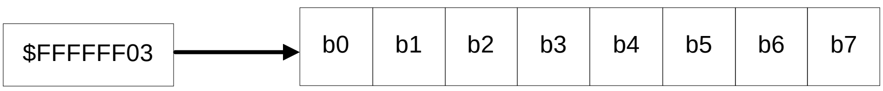
28. Assume two eight bit registers available at address FFFFFF03h and FFFFFFF05h in memory space. The first register, called DATA, holds a character supplied by an external device. The second register STATUS has a single read-only "sticky-bit" *RxRdy* which is set (1) each time the data register is filled with a new value The bit is reset (0) by the peripheral device when the data register is read. Remaining bits in this registers are always read as 0. Write a:

```
      procedure ReadRegister ( valid : out BOOLEAN; rdata :out BYTE)
```

That either returns with "fresh" data (valid=TRUE) or "old" data (valid=FALSE).

Interrupts in Ada95

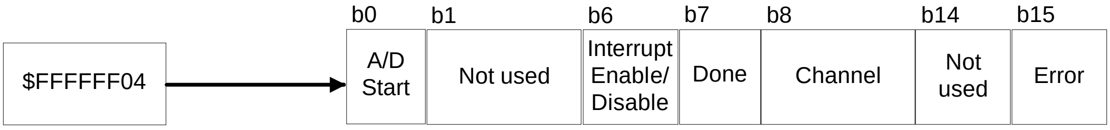
29. Assume an eight bit data register `DATA` available at address FFFFFF03h in memory space. Show an interrupt handler (protected object `Interrupt`) with procedure `Handler` and entry `Read`. In `Handler` the register is read and the value is placed in a local variable. The local variable value, presumed that `Handler` was activated, is returned by the entry `Read`, within the interrupt handler.



a) Write a protected object with a *static* (hardware priority) interrupt handler.
b) Write a protected object with a *dynamic* (hardware priority) interrupt handler.

30. Assume an eight bit register available at address FFFFFF04h in memory space (se below).

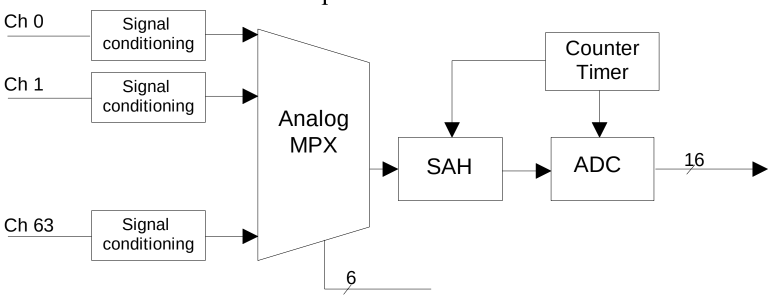
a) Define an appropriate type for this register layout.
b) Write a package `Cntrl_Reg` with functions `Read_Done`, returning the *Done*-bit, and `Read_Error` returning the *Error*-bit.
c) Add to the package a procedure `Write_Reg(FLAG:FLAG:CHAN_TYPE)` that updates the fields: *A/D Start*, *Interrupt Enable/Disable* and *Channel* simultaneously. Use value 0 for the read only bits *Done* and *Error*.



d)
The register is a control register for an A/D converter where bits have the following functions:

A/D Start	Set to 1 to initiate conversion.
Interrupt Enable/Disable	Set to 1 if the finalized conversion should generate an interrupt
Done	If this bit is zero, the conversion is still pending, otherwise it's ready
Channel	Chooses one out of the 64 analog inputs.
Error	This bit is clear if the conversion was ok, otherwise it is set.

When a conversion is initiated the converter samples the value from *Channel.*



The *Done* bit is reset. The sampled value is converted to a binary value and placed in a 16 bit data register located at FFFFFF02. The *Done* bit is now set. If the *Interrupt Enable/Disable* is set an interrupt is generated. Write an ADA package `AD_Converter` with a single procedure, using *Interrupt Enable*:

```
procedure Read_AD(Ch: in CHAN_TYPE; M:out MEASUREMENT ;
      AD_busy: out BOOLEAN);
```

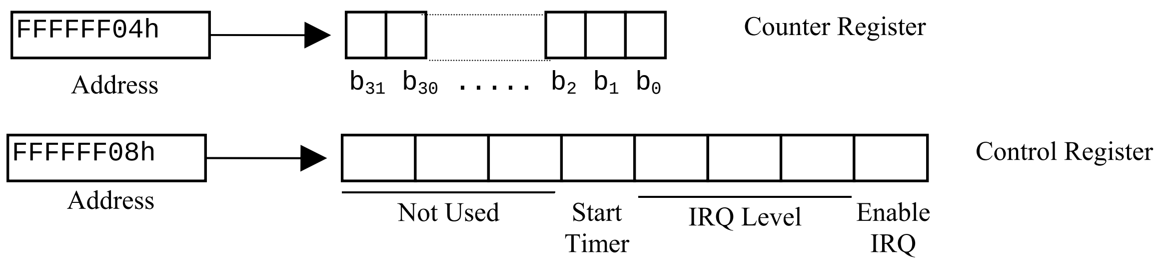
If the conversion was succesful, then `M` holds the converted value and `AD_Busy` is FALSE. If the converter is busy, then `AD_Busy` is TRUE and `M` is undefined. If a conversion error occurs, then exception `Conversion_Error` should be raised.

-- följande är en näst intill obegriplig formulering av uppgift, men bra exempel på när requeue måste användas. detta måste formuleras om väsentligt.

31. Implementera ett Ada-paket som tillhandahåller proceduren `Current_Temp(degrees: out integer)` vilken returnerar nuvarande temperatur via en inkopplad temperaturenhet.

Det finns två procedurer du kan använda:
Proceduren *init_temp* gör att temperaturenheten läser temperaturen och genererar ett avbrott när den är klar. Avbrottets identifierare är PORTBINT och det sker på avbrottsnivå 4. *temp_device(degrees: out integer)* läser av temperaturenheten via en I/O-port.

32. A timer counter device has the following registers:

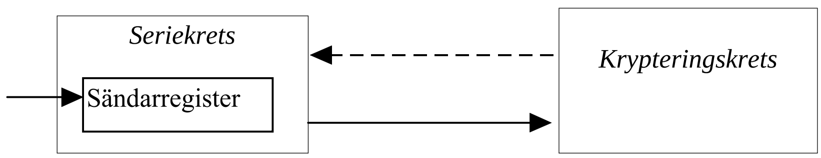


Control register bits have the following meanings:

Start Timer	1 starts the counter, 0 stops the counter.
IRQ Level	Hardware interrupt level, 001 is lowest, 111 is highest level.
Enable IRQ	1 means device request interrupt each time counter reach zero.

When the timer is started, the counter will count down the value in the counter register by one for each clock cycle. System clock is 20 MHz. When the counter reaches zero, the initial value of the counter register (as when the timer was started) is reloaded and the countdown starts all over again.
The hardware interrupt level is assigned 6 for this device. The interrupt period should be 1 ms.
Write a protected object which specifies the interrupt procedure `Handler` and the protected procedure `InitTimer`. Also show the bodies. It is assumed that the `Handler` doesn't do anything yet.

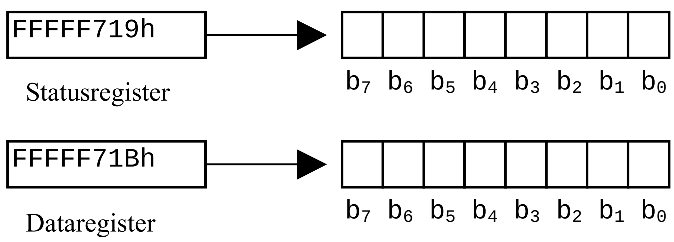
33.



En extern krypteringskrets är inkopplad till ett datorsystem enl. figur. Tecken som sänds till krypteringskretsen krypteras automatiskt, och sänds tillbaka i krypterad form. För kommunikationen till och från krypteringskretsen används en *seriekommunikationskrets*.

a) Implementera ett ada-program som hämtar tecken från en inbuffert *inbuf* och skickar dessa till krypteringskretsen. Bufferten finns tillgänglig i paketet inbuf och fylls kontinuerligt med tecken från ett tangentbord. I denna uppgift ignoreras de krypterade tecken som kommer i retur.

Specifikationen för inbuf är:
```
package inbuf is
      procedure get(c:character);
end;
```
Seriekommunikationskretsen har ett status- och ett dataregister. Statusregistret har address FFFFF719h och dataregistret address FFFFF71Bh, se nedan. Sändning påbörjas så snart ett nytt tecken skrivit till dataregistret. Bit b2 i statusregistret är 1 då kretsen har avslutat sändning och är beredd att skicka ett nytt tecken.



b) Utöka lösningen ovan så att de krypterade tecknen tas emot och läggs i en buffert genom anropet utbuf.put(c: in character). Seriekretsen signalerar mottaget tecken på avbrottet portAint med avbrottsprioritet 105. Det mottagna tecknet finns då att läsa i kretsens dataregister.

Interfacing Ada95 to C and assembly language

34. Show the "skeleton" of an interrupt handler routine in CPU32 assembly language. Assume that the hardware registers A0, A1, D0 and D1 are used by the handler.

35. An assembly routine manipulating the processor interrupt level is specified in 'C' as follows:

```
int   asm_spl( int new_level )
```

This routine sets the new processor interrupt level to `new_level`, and returns the previous interrupt level.

a) Show the assembly routine in CPU32 assembly language.
b) Show how to import `asm_spl` to a function `spl` in an Ada-program.
EXAMPLE USE:

```
Priority:   integer;
…
Priority := spl( 7 );
-- Critical (uninterruptible) region
Priority :=  spl( Priority );       -- restore priority
```

36. Assume that the following declaration specifies a function from a 'C'-library.

```
void  Cgrow( int item, int amount );
```

Show how to import this C-function (as procedure `grow` into an Ada program.

37. Assume that the following declaration specifies a function from a 'C'-library.

```
int   Cfuzzyval( int v1, long v2 );
```

Show how to import this C-function (as procedure `fuzzyval` into an Ada program.

38. Assume that the following declarations specifies a data structure and a function from a 'C'-library.

```
struct Timeval {
      long  tv_sec;
      long  tv_usec;
};

int SetIntervalTimer ( struct Timeval * );
```

a) Show how to import these into an Ada program.
b) Show how to call the function 'SetIntervalTimer' from an Ada program.

39. Assume an Ada procedure as follows:

```
procedure   Do_Something;
```

Show how this procedure can be declared as a function `C_do_something` and then make it accessible from a C-program.

Worst Case Execution Time estimation

40. Consider a processor clocked at 100 MHz. Assume that there are instructions that can be executed during a clock cycle. State the least possible"time unit" that can be expressed as an integer and also represent execution of every instruction.

41. Consider the following code fragment. Time is supposed to be an integer of type "time unit" and the numbers within parenthesizes denote the calculated WCET for the line.

```
...
1:    (4)   :     a := b + c;
2:    (5)   :     if( a < 9 ) then
3:    (64)  :           S1;
4:    (5)   :     else if a < 100 then
5:    (112) :           S2;
6:    (2)   :     else
7:    (112) :           S3;
8:    (1)   :     end;
...
```

a) State the possible paths (using line indexes) through the code.
b) Using Shaw's method, estimate WCET for the possible paths.

42. Consider the procedure `Main` below. Assume that:

- The cost for *assignment*, *return* and *comparison* is one time unit.
- A *function call* overhead is one time unit.
- *Addition* and *subtraction* costs are two time units.
- Other language constructs will not generate any code so they are "null" cost.

a) Using Shaw's method, estimate WCET for the procedure.
b) Now, suppose the value of `A` was undetermined in `Main`. How would you then try to estimate WCET?

```
procedure Main is
      A : Natural := 4;
      F : Natural;

      function Calculate (Z : in Natural) return Natural is
            R : Natural;
            begin
                  if Z == 0 then
                        R := 1;
                  else if Z == 1 then
                        R := 1;
                  else
                        R := Calculate(Z-1) + Calculate(Z-2);
                  end;
            return R;
      end Calculate;

begin
      F := Calculate(A);
end;
```

43. Consider the function `Calculate` below. Assume that:

- The cost for *assignment*, *return* and *comparison* is one time unit.
- A *function call* overhead is one time unit.
- *Addition* and *subtraction* costs are two time units.
- *Multiplication* and *division* costs are five time units.
- Other language constructs will not generate any code so they are "null" cost.

Using Shaw's method, estimate WCET for the function.

```
type Matrix is array (1..8,1..8) of Natural;

function Calculate(M : in Matrix) return Natural is
      F : Natural := 0;
      C : Natural := 0;

      function Select(X, Y, Z : in Natural) return Natural is
            R : Natural;
      begin
            if Y < Z then
                  R := Y;
            else
            if X < Z then
                  R := Z;
            else
                  R := X;
            return R;
      end Select;

begin
      for i in 1 .. 6 loop
            for j in i+1 .. 7 loop
                  F := F + Select(M(i,j-1),M(i,j),M(i,j+1));
                  C := C + 1:
            end loop;
      end loop;
      return F / C;
end Calculate;
```

Processor utilisation analysis

44. Consider the following task set:

Task	Period *T* [ms]	Deadline *D* [ms]	Execution time *C* [ms]
A	100	100	22
B	50	50	10
C	25	25	8

a) Calculate LCM (least common multiple) for the set.
b) Calculate the processor utilization factor.

45. Suppose conditions for RMSA is met by the following task set. Show if or if possibly not, the task set is schedulable.

Task	Period *T* [ms]	Deadline *D* [ms]	Execution time *C* [ms]	priority
A	7	7	1	0
B	14	14	1	1
C	18	18	4	2

Response time analysis

46. Suppose conditions for RMSA is met by the following task set. Show that:
   a) This set is NOT necesserily schedulable due to RMSA?
   b) It is, in fact, schedulable, due to response time analysis.

Task	Period *T* [ms]	Deadline *D* [ms]	Execution time *C* [ms]
A	30	30	10
B	40	40	10
C	60	60	14

47. Consider the following task set:

Task	Period *T* [ms]	Deadline *D* [ms]	Execution time *C* [ms]
A	10	7	3
B	12	6	4

a) Determine if the set is schedulable due to Rate Monotonic priority assignment.
b) Determine if the set is schedulable due to Deadline Monotonic priority assignment.
A full motivation is required.

48. Consider the following task set:

Task	Period *T* [ms]	Deadline *D* [ms]	Execution time *C* [ms]
A	70	65	15
B	40	40	10
C	30	12	10

Assign priorities according to deadline monotonic and then, does every task within this set meet its deadline?

49. Consider the following task set:

Task	Period *T* [ms]	Deadline *D* [ms]	Execution time *C* [ms]
A	1000	20	3
B	100	100	10
C	50	50	20
D	57	10	5
E	33	33	1
F	7	7	1

a) Calculate processor utilization for the set.
b) Give the execution order for “rate monotonic” priority assignment.
c) Give the execution order for “deadline monotonic” priority assignment.
d) State maximum response times for “deadline monotonic” priority assignment.
e) Will all deadlines be met?
f) State maximum response times for “rate monotonic” priority assignment.
g) Will all deadlines be met?
h) Add another task “FT” with period 30, deadline 5, execution time (WCET) 2. Assuming “deadline monotonic” priority assignment, will all deadlines be met?

50. Consider the following three tasks:

Task	Period *T* [ms]	Deadline *D* [ms]	Execution time *C* [ms]	priority
A	50	10	5	1
B	500	500	240	2
C	3000	3000	1000	3

The tasks utilize semaphores, s_1, s_2 and s_3 as follows:

Task	***Usage***
A	*lock(s_1); unlock(s_1);*
B	*lock(s_2); lock(s_3); unlock(s_3); unlock(s_2);*
C	*lock(s_3); lock(s_2); unlock(s_2); unlock(s_3);*

The critical sections lengths are as follows:

A uses s_1 max 5 ms	$cs_{P1,S1}$ = 5 ms
B uses s_2 max 10 ms	$cs_{P2,S2}$ = 10 ms
B uses s_3 max 5 ms	$cs_{P2,S3}$ = 5 ms
C uses s_2 max 10 ms	$cs_{P3,S2}$ = 10 ms
C uses s_3 max 25 ms	$cs_{P3,S3}$ = 25 ms

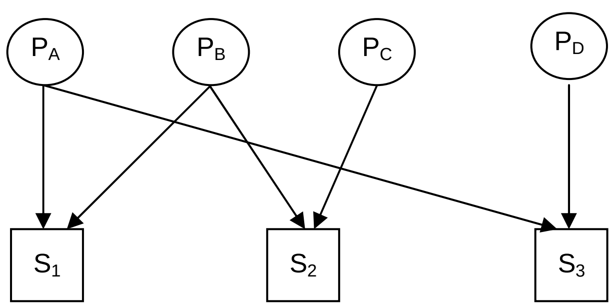
Note that time $cs_{P2,S3}$ is included in $cs_{P2,S2}$.

Assume that ICPP (Immediate Ceiling Priority Protocol) is used to lock and unlock the semaphores. Is this task set then schedulable?

51. The following task set should be scheduled due to *deadline monotonic*.
Three semaphores; S_1, S_2 and S_3 are used to synchronize the tasks.
H_{Si} denotes the maximum locking time when a task locks semaphore *i*.
P_A, P_B, P_C and P_D denote the tasks A,B,C and D.

Task	*T* [ms]	*D* [ms]	*C* [ms]	Priority	H_{S1} [ms]	H_{S2} [ms]	H_{S3} [ms]
A	5	4	2	1	1	-	1
B	16	12	3	2	1	2	-
C	20	16	3	3	-	3	-
D	28	28	4	4	-	-	2

The following access graph illustrates use of the semaphores.



Assume that ICPP (Immediate Ceiling Priority Protocol) is used to lock and unlock the semaphores. Is this task set then schedulable?

52. Consider the following task set:

Task	Period *T* [ms]	Deadline *D* [ms]	Execution time *C* [ms]
A	7	7	2
B	8	8	1
C	10	10	3
D	25	25	2

Priorities are assigned due to *deadline monotonic*.

a) Show that a processor utilization test cannot guarantee that this set is schedulable.
b) Determine response times due to response time analysis.
c) Add two semaphores *S* and *Q*. Assume ICPP and the following:
Task B and task C uses *S*, task C and task D uses *Q*, *critical section S*=1 ms and *critical section Q* = 2ms
Determine response times due to response time analysis considering these semaphores.

53. Determine the maximum number of bits in the following CAN-messages:
   a) 2 bytes in standard CAN
   b) 8 bytes in standard CAN
   c) 3 bytes in EXTENDED CAN
   d) 7 bytes in EXTENDED CAN.

54. The following message set shall be scheduled on a CAN-bus at bitrate 50 Kbit/s:

Message	*T* [ms]	*D* [ms]	*# of bytes*
M1	10	4	1
M2	20	6	5
M3	15	10	6

Assign priorities due to deadline monotonic and:
   a) Determine response times.
   b) Calculate the bus utilization factor.

Processor demand analysis

55. The following task set should be scheduled due to *earliest deadline first* (EDF).

Task	*T* [ms]	*D* [ms]	*C* [ms]
A	3	2	1
B	4	2	1
C	5	4	2

a) Calculate processor utilization factor.
b) Draw a timing diagram showing a possible scenario for execution order. ("simulation").
c) Determine if the task set can be scheduled by performing processor demand analysis.

56. A real-time system with three periodic tasks is scheduled due to EDF. The run-time system executes all tasks with preemption and due to task priorities. The following table details periods (*T*), deadlines (*D*) and worst case execution times (*C*). All tasks arrives at *t* = 0.

Task	*T* [ms]	*D* [ms]	*C* [ms]
A	4	4	3
B	10	4	1
C	20	16	3

a) Show that Liu & Layland's simple utilization based test is inapplicable in this case.
b) Use *processor demand analysis* to determine whether the task set is schedulable or not

57. A real-time system with five periodic tasks is scheduled due to EDF. The run-time system executes all tasks with preemption and due to task priorities. The following table details periods (*T*), deadlines (*D*) and worst case execution times (*C*). All tasks arrives at *t* = 0.

Task(P)	*T* [ms]	*D* [ms]	*C* [ms]
1	6	6	2
2	15	15	2
3	16	16	4
4	10	10	2
5	15	15	1

a) Use processor demand analysis and show that this task set is schedulable due to EDF.
b) Draw a timing diagram (*t* = 0 – 32) showing how the tasks will be executed due to EDF.

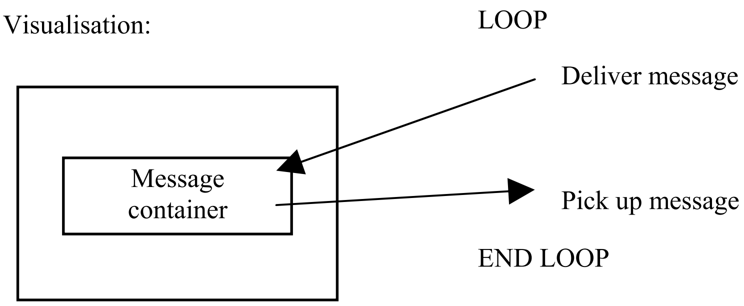
58. A real-time system with three periodic tasks is scheduled due to EDF. The run-time system executes all tasks with preemption and due to task priorities. The following table details periods (*T*), deadlines (*D*) and worst case execution times (*C*). All tasks arrives at *t* = 0.

Task	*T* [ms]	*D* [ms]	*C* [ms]
A	4	4	3
B	20	18	2
C	10	3	1

a) Use processor demand analysis and show that this task set is schedulable due to EDF.
b) Draw a timing diagram (*t* = 0 – *LCM(A, B, C)*) showing how the tasks will be executed due to EDF.

Solutions: Parallel programming in Ada95

1. Visualisation:



The specification file (.ads) :

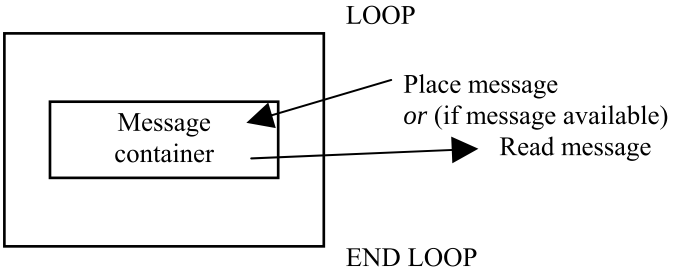
```
--
-- mailbox_handler.ads
-- Assignment 1
package MAILBOX_HANDLER is

type MSG_TYPE is new character;

task  MAILBOX is
   entry    PUT_MESSAGE( msg : in MSG_TYPE );
   entry    GET_MESSAGE( msg : out MSG_TYPE );
end MAILBOX;

end MAILBOX_HANDLER;
```

The body file (.adb):

```
--
-- mailbox_handler.adb
-- Assignment 1

package body MAILBOX_HANDLER is

task  body MAILBOX is
      contents    : MSG_TYPE;
begin
      loop
            accept PUT_MESSAGE( msg : in MSG_TYPE ) do
            contents := msg;
            end PUT_MESSAGE;

            accept GET_MESSAGE( msg : out MSG_TYPE ) do
                msg := contents;
            end GET_MESSAGE;

      end loop;
   end MAILBOX;

end MAILBOX_HANDLER;
```

Note: Accept statements are executed sequentially which guarantees that entry calls can only be serviced as: PUT, GET, PUT, GET etc. Codes within do-end in accept statements are critical regions, which guarantees that the variable only can be accessed by either a writer or a reader at the same time.

2.




```
-- notice_handler_1.ads
-- Assignment 2
package NOTICE_HANDLER_1 is

type MSG_TYPE is new character;

task  MAILBOX is
   entry    PUT_MESSAGE( msg : in MSG_TYPE );
   entry    GET_MESSAGE( msg : out MSG_TYPE );
end MAILBOX;

end NOTICE_HANDLER_1;

-- notice_handler_1.adb
-- Assignment 2

package body NOTICE_HANDLER_1 is

task  body MAILBOX is
      contents    : MSG_TYPE;
      msg_available : boolean := false;
begin
      loop
         select
            accept PUT_MESSAGE( msg : in MSG_TYPE ) do
            contents := msg;
            end PUT_MESSAGE;
            msg_available := true;

         or
              when msg_available =>
               accept GET_MESSAGE( msg : out MSG_TYPE ) do
                  msg := contents;
               end GET_MESSAGE;
         end select;

   end loop;
   end MAILBOX;

end NOTICE_HANDLER_1;
```

Note the “select”-statement, which makes it possible to choose to accept either a call to PUT or a call to GET. If there are queued tasks waiting for both PUT and GET, then one of them is arbitrarily selected for rendezvous.

3.
```
-- notice_handler_2.ads

package NOTICE_HANDLER_2 is

type MSG_TYPE is new character;

task  MAILBOX is
   entry    PUT_MESSAGE( msg : in MSG_TYPE );
   entry    GET_MESSAGE( msg : out MSG_TYPE );
end MAILBOX;

end NOTICE_HANDLER_2;


-- notice_handler_2.adb

package body NOTICE_HANDLER_2 is

task  body MAILBOX is
      contents    : MSG_TYPE;
      msg_available : boolean := false;
begin
      loop
         select
            accept PUT_MESSAGE( msg : in MSG_TYPE ) do
            contents := msg;
            end PUT_MESSAGE;
            msg_available := true;

         or
              when msg_available and PUT_MESSAGE'COUNT = 0 =>
               accept GET_MESSAGE( msg : out MSG_TYPE ) do
                  msg := contents;
               end GET_MESSAGE;
               -- Uncomment following line to avoid duplicates ...
               --msg_available := false;
         end select;

   end loop;
   end MAILBOX;

end NOTICE_HANDLER_2;
```

4.

a)

```
-- Task specification
task BOARD is
      entry PUT_MESSAGE(msg : in MSG_TYPE);
      entry GET_MESSAGE( msg : out MSG_TYPE);
end BOARD;

task body BOARD is
      message : MSG_TYPE;
      msg_available : boolean := false;

begin
      loop
            select
                  accept PUT_MESSAGE( msg: in MSG_TYPE) do
                        message := msg;
                  end PUT_MESSAGE;
                  msg_available := true;

            or
                  when msg_available and PUT_MESSAGE'COUNT= 0 =>
                        accept GET_MESSAGE( msg: out MSG_TYPE) do
                              msg := message;
                        end GET_MESSAGE;

            or
                  delay 5;
                  msg_available := false;

            end select;
      end loop;
end BOARD;
```

← The select-statement is completed with this alternative.

5.

```
task body CONSUMER is
      ...
      ...
      msg: MSG_TYPE;
      ...
      ...

begin
      loop
            WORK;

            select
                  BOARD.GET_MESSAGE( msg );
                  HANDLE_MESSAGE( msg );

            else
                  null;
            end select;
      end loop
end CONSUMER;
```

6.

```
procedure ASSIGNMENT_6 is
      type MSG_TYPE is .....

task type PROD is
      entry Init(Id: in integer);
end PROD;

task type MAILBOX is
      entry PUT_MESSAGE(Msg : in MSG_TYPE );
      entry GET_MESSAGE(Msg : out MSG_TYPE);
end MAILBOX;

task CONSUMER;

N: constant integer := 20;
Producers: array (1..N) of PROD;
Mailboxes: array (1..N) of MAILBOX;
```

We can reuse results from assignment 2 for implementation of `GET_MESSAGE` and `PUT_MESSAGE`. We may write code for producer and consumer.

```
task body PROD is
      Message: MSG_TYPE;
      My_Id : integer;
begin
      accept Init(Id: in integer) do
            My_Id := Id;
      end Init;

      loop
            -- Create a message
            Mailboxes(My_Id).PUT_MESSAGE(Message);
      end loop;
end PROD;

task body MAILBOX is -- As in Assignment 1  ....

task body CONSUMER is
      Number_Empty: integer;
      Mess: MSG_TYPE;
begin
      loop
            Number_Empty := 0;
            for I in 1..N loop
                  select
                        Mailboxes(I).GET_MESSAGE(Mess);
                        Ta_hand_om_brev(Mess);
                  else
                        Number_Empty := Number_Empty + 1;
                  end select;
            end loop;
            if Number_Empty = N then
                  delay 20.0;
            end if;
      end loop;
end CONSUMER;

begin
      for I in 1..N loop
            Producers(I).Init(I);
      end loop;
end Uppgift_7;
```

7.

```
protected BOARD is
      procedure PUT_MESSAGE(msg : in MSG_TYPE);
      entry GET_MESSAGE(msg : out MSG_TYPE);
private
      Message : MSG_TYPE;
      Msg_Available : boolean := false;
end BOARD;

protected body BOARD is
      procedure PUT_MESSAGE(msg : in MSG_TYPE) is
      begin
            Message := msg;
            Msg_Available := true;
      end PUT_MESSAGE;
      entry GET_MESSAGE(msg : out MSG_TYPE)
            when Msg_Available is
      begin
            msg := Message;
      end GET_MESSAGE;
end BOARD;
```

8.

```
package Notice_Board is
   type Var_Num is range 1 .. 3;
   procedure Read( Num : in Var_Num; Value : out Integer);
   procedure Write( Num : in Var_Num; Value : in Integer);
end Notice_Board;

package body Notice_Board is
   protected type Protected_Int is
      entry Read( Value : out Integer);
      procedure Write( Value : in Integer);
   private
      X : Integer := 0;
      Written : Boolean := False;
   end Protected_Int;

   protected body Protected_Int is
      entry Read(Value : out Integer) when Written is
      begin
         Value := X;
      end;

      procedure Write(Value : in Integer) is
      begin
         X := Value;
         Written :=  True;
      end;
   end Protected_Int;

   type Protected_Int_List is array (Var_Num) of Protected_Int;
   Board_Variables : Protected_Int_List;

   procedure Read( Num : in Var_Num; Value : out Integer) is
   begin
      Board_Variables(Num).Read(Value);
   end;

   procedure Write ( Num : in Var_Num; Value : in Integer) is
   begin
      Board_Variables(Num).Write(Value);
   end;
```

```
end Notice_Board;
```

9.

a)

```
procedure Traffic is

   task Junction is
      entry Take;
      entry Release;
   end Junction;

   task body Junction is
      Busy : Boolean := False;
   begin
      loop
         select
            when Busy=False =>
            accept Take do
               Busy := True;
            end Take;
         or
            accept Release do

               Busy := False;
            end Release;
         end select;
      end loop;
   end Junction;

   -- Client tasks switching lights...!
begin
   null;
end Traffic;
```

b)

```
procedure Traffic is

   protected Junction is
      entry Take;
      procedure Release;
   private
      Busy : Boolean := False;
   end Junction;

   protected body Junction is
      entry Take when not Busy is
      begin
         Busy := True;
      end Take;

      procedure Release is
      begin
         Busy := False;
      end Release;
   end Junction;

   -- Client tasks switching lights...!
begin
   null;
end Traffic;
```

10.

```
package resource is
    type res_range is range 1..10;
    procedure acquire(nr_of_resources : in res_range);
    procedure tryto_acquire(nr_of_resources : in res_range; ok :
                            out boolean);
    procedure release(nr_of_resources : in res_range);
end resource;

package body resource is
    protected res_handler is
         entry allocate(nr_of_res : in res_range);
         procedure tryto_allocate(nr_of_res : in res_range;
                                  ok : out boolean);
         procedure release(nr_of_res : in res_range);
    private
         entry assign(nr_of_res : in res_range);
         available : res_range := res_range'last;
         new_resources_released : boolean := false;
         nr_to_check : natural := 0;
    end res_handler;

    protected body res_handler is
    entry allocate(nr_of_res : in res_range) when available > 0 is
    begin
         if nr_of_res <= available then
              available := available - nr_of_res;
        else
              requeue assign;
         end if;
    end allocate;

    entry assign(nr_of_res : in res_range) when
                  new_resources_released is
    begin
         nr_to_check := nr_to_check - 1;
         if nr_to_check = 0 then
              new_resources_released := false;
         end if;
         if nr_of_res <= available then
              available := available - nr_of_res;
         else
              requeue assign;
        end if;
    end assign;

    procedure tryto_allocate(nr_of_res : in res_range;
                             ok : out boolean) is
    begin
         if nr_of_res <= available then
              available := available - nr_of_res;
              ok := true;
         else
              ok := false;
         end if;
    end tryto_allocate;

    procedure release(nr_of_res : in res_range) is
    begin
         available := available + nr_of_res;
         if assign'count > 0 then
              nr_to_check := assign'count;
              new_resources_released := true;
         end if;
    end release;

    end res_handler;

    procedure acquire(nr_of_resources : in res_range) is
    begin
         res_handler.allocate(nr_of_resources);
    end acquire;
```

```
        procedure tryto_acquire(nr_of_resources : in res_range;
                                ok : out boolean) is
        begin
             res_handler.tryto_allocate(nr_of_resources, ok);
        end tryto_acquire;

        procedure release(nr_of_resources : in res_range) is
        begin
             res_handler.release(nr_of_resources);
        end release;

    end resource;
```

11.

```
type Msgtype is -- something

monitor Board is
   procedure Putmsg(Msg : in Msgtype);
   procedure Getmsg(Msg : out Msgtype);
   Contents : Msgtype;
   Empty : Boolean := True;
   cond_contents : Condition_Variable;
   waiting: integer:=0;
end Board;

monitor body Board is

procedure Putmsg(Msg : in Msgtype) is
begin
   Contents := Msg;
   Empty := False;
   if waiting>0 then
         waiting:=waiting-1;
         Signal(Cond_contents);
   end if;
end Putmsg;

procedure Getmsg(Msg : out Msgtype) is
begin
   if not Empty then
      Msg := Contents;
   else
      waiting:=waiting+1;
      Wait(Cond_contents);
      Msg := Contents;
      if waiting>0 then
        waiting:=waiting-1;
        Signal(Cond_contents);
      end if;
   end if;
end Getmsg;

end Board;
```

12.

```
type Msgtype is -- something

package Board is
   procedure Putmsg(Msg : in Msgtype);
   procedure Getmsg(Msg : out Msgtype);
end Board;

package body Board is
   Contents : Msgtype;
   Empty : Boolean := True;
   sem_board, Sem_contents: Semaphore;
   waiting: integer:=0;
begin

   procedure Putmsg(Msg : in Msgtype) is
   begin
      wait(sem_board);
      Contents := Msg;
      Empty := False;
      while waiting > 0 loop
         signal(Sem_contents);
         waiting:=waiting-1;
      end loop;
      signal(sem_board);
   end Putmsg;

   procedure Getmsg(Msg : out Msgtype) is
   begin
      wait(sem_board);
      if not Empty then
         Msg := Contents;
      else
         waiting:=waiting+1;
         signal(sem_board);
         wait(Sem_contents);
         wait(sem_board);
         Msg := Contents;
      end if;
      signal(sem_board);
   end Getmsg;

end Board;
```

13.

a)

```
package body Hairy is

   monitor Synchronize is
      procedure Arrival;
      procedure Removal;
      procedure Start_Cut;
      procedure Cutfinished;
      Shaggy, Smart: Condition_Variable;
      On_Conveyor: Integer;
   end Synchronize;

   monitor body Synchronize is

      procedure Arrival is
      begin
         if On_Conveyor=0 then
            On_Conveyor:=1;
            Send(Shaggy)
         else
            On_Conveyor:=On_Conveyor+1;
         end Arrival;

         procedure Removal is
         begin
            Wait(Smart);
         end Removal;

         procedure Start_Cut is
         begin
            if On_Conveyor=0 then
               Wait(Shaggy)
            end if;
            On_Conveyor:=On_Conveyor-1;
         end;

         procedure Cutfinished is
         begin
            Send(Smart);
         end Cutfinished;
      end Synchronize;

   end Hairy;
```

b)

```
package body Hairy is

   Shaggy, Smart: semaphore:=0;

   procedure Arrival is
   begin
      Signal(Shaggy)
   end Arrival;

   procedure Removal is
   begin
      Wait(Smart);
   end Removal;

   procedure Start_Cut is
   begin
      Wait(Shaggy)
   end;

   procedure Cutfinished is
   begin
      Signal(Smart);
   end Cutfinished;


end Hairy;
```

14.

a)

```
package body SafeControl is

   monitor SafeCommands is
      procedure Start;
      procedure Close;
      Cond_Closed: Condition_Variable;
   end SafeCommands;

   monitor body SafeCommands is

      procedure Start is
      begin
         if not is_closed then
            Wait(Cond_Closed);
         end if;
         start_vehicle;
      end Start;

      procedure Close is
      begin
         close_doors;
         Signal( Cond_Closed );
      end Close;
   end SafeCommands;

end SafeControl;
```

b)

```
package body SafeControl is
   Sem_Closed: Semaphore:=0;  -- receives signal when doors are closed
   Sem_Ioaccess: Semaphore:=1; -- for mutual exclusion

   procedure Start is
   begin
      wait( Sem_Closed );     -- doors must be closed
      wait( Sem_Ioaccess );   -- mutex..
      start_vehicle;
      signal( Sem_Ioaccess );
   end Starta;

   procedure Close is
   begin
      wait( Sem_Ioaccess );
      close_doors;
      signal( Sem_Ioaccess );
      signal( Sem_Closed );
   end Close;

end SafeControl;
```

15. To be demonstrated at exercise.

16.
a)
```
procedure Blockerad_Filosof is

   Max:constant Integer:=5;

   subtype Filint is Integer range 1..Max;
   type Free_Array is array(Filint) of Boolean;

   protected Bord is
      entry Getsticks(I:in Filint);
      procedure Releasesticks(I:in Filint);

   private
      entry Waitforsticks(I:in Filint);
      Free : Free_Array := (others => True);
      Stickcheck: Integer:=0;
   end;

   task type Filosof is
      entry Start(Id:in Filint);
   end;

   Filosofer:array(Filint)of Filosof;

   protected body Bord is

      entry Getsticks(I:in Filint) when True is
      begin
         if  Free(I) and Free((I mod Max)+1) then
            Free(I):=False;
            Free((I mod Max)+1):=False;
         else
            requeue Waitforsticks;
         end if;
      end Getsticks;

      entry Waitforsticks(I:in Filint) when Stickcheck>0 is
      begin
         Stickcheck:=Stickcheck-1;
         if  Free(I) and Free((I mod Max)+1) then
            Free(I):=False;
            Free((I mod Max)+1):=False;
         else
            requeue Waitforsticks;
         end if;
      end Waitforsticks;

      procedure Releasesticks(I:in Filint) is
      begin
         Free(I):=True;
         Free((I mod Max)+1):=True;
         Stickcheck:=Waitforsticks'Count;
      end Releasesticks;
   end Bord;

   task body Filosof is

      Myid:Filint;
      Ok:Boolean := False;

      procedure Think is
      begin
```

```
         delay 3.0;
      end Think;

      procedure Eat is
      begin
         delay 2.0;
      end;

   begin
      accept Start(Id:in Filint) do
         Myid:=Id;
      end Start;
      loop
         Think;
         Bord.Getsticks(Myid);
         Eat;
         Bord.Releasesticks(Myid);
      end loop;
   end Filosof;

begin
   for I in Filint loop
      Filosofer(I).Start(I);
   end loop;
end Blockerad_Filosof;
```

b)

If only a single tothpick is required in each request, a simple guard `when tothpics>0` could have been used.

In this case, the resources requested are always two distinct tothpicks which complicates this issue since we need to know the actual philosopher requesting toothpicks to desire whether they are both available. But ”the actual philosopher” is an entry parameter, and cannot be used in evaluating the guard. Thus, we must allow entrance to the entry (mutual exclusion) and then desire whether we can acknowledge the request or not. If we can’t we has to requeue this request.

17.

a)

```
procedure Bridge_Proc is
   Groupsize: constant Integer:=10;

   task Bridge is
      entry Request_W;
      entry Request_E;
      entry Release;
   end Bridge;

   task body Bridge is
      Free: Boolean := True;
      At_West, At_East: Integer:=0; --People from the granted group
still waiting

   begin
      loop
         select
            when (Free and Request_W'Count=Groupsize) or At_West>0 =>
            accept Request_W do
               null;
            end Request_W;
            if Free then
               Free:=False;
               At_West:=Groupsize;
            end if;
            At_West:=At_West-1;
         or
            when (Free and Request_W'Count=Groupsize) or At_East>0 =>
            accept Request_E do
               null;
            end Request_E;
            if Free then
               Free:= False;
               At_East:=Groupsize;
            end if;
            At_East:=At_East-1;
         or
            accept Release do
               null
            end Release;
            Free:=True;
         end select;
      end loop;
   end Bridge;

begin
   null;
end Bridge_Proc;
```

b)

```
procedure Bridge_Proc is
   Groupsize: constant Integer:=10;

   protected Bridge is
      entry Request_W;
      entry Request_E;
      procedure Release;
   private
      Free: Boolean := True;
      At_West, At_East: Integer:=0;
   end Bridge;

   protected body Bridge is

      entry Request_W when
            (Free and Request_W'Count=Groupsize) or At_West>0 is
      begin
         if Free then
            Free:= False;
            At_West:=Groupsize;
         end if;
         At_West:=At_West-1;
      end Request_W;

      entry Request_E when (Free and Request_W'Count=Groupsize) or
At_East>0 is
      begin
         if Free then
            Free:= False;
            At_East:=Groupsize;
         end if;
         At_East:=At_East-1;
      end Request_E;


      procedure Release is
      begin
         Free:=True;
      end Release;
   end Bridge;

begin
   null;
end Bridge_Proc;
```

18. TODO!

Implement a protected objet. The object shall manage a circular buffer holding 8 items of type Data. The object shall have two entries:

Entry PUT is used to add items to the buffer. If there is insufficient room, the caller shall be blocked until the buffer can accept the request.
Entry GET is used to pick items from the buffer. If the number of items in the buffer is to small to fulfill a request, the caller shall be blocked until the request can be granted.

19.

a)

```
procedure Robotsa is
   subtype Id_Type is Integer;
   Capacity: Integer:=100;
   Detail_Count: Integer:=0;
   type Detail_Type is record
      Id: Integer:=0;
   end record;
   type Details_Type is array (1..Capacity) of Detail_Type;
   protected Mailbox is
      entry Put_Detail(Detail: in Detail_Type);
      entry Get_Detail(Detail : out Detail_Type);
   private
      Details: Details_Type;
      I, Interested, Details_Count: Integer:=0;
      New_Detail: Boolean := False;
   end Mailbox;
   protected body Mailbox is
      entry Put_Detail(Detail: in Detail_Type)
         when Detail_Count<Capacity is
      begin
         Detail_Count:=Detail_Count+1;
         Details(Detail_Count):=Detail;
      end Put_Detail;
      entry Get_Detail(Detail : out Detail_Type)
         when Detail_Count>0 is
      begin
         Detail:=Details(Detail_Count);
         Detail_Count:=Detail_Count-1;
      end Get_Detail;
   end Mailbox;
begin
   null;
end Robotsa;
```

b)

```
procedure Robotsb is
   subtype Id_Type is Integer;
   Capacity: Integer:=100;
   Detail_Count: Integer:=0;
   type Detail_Type is record
      Id, Typ: Integer:=0;
   end record;
   type Details_Type is array (1..Capacity) of Detail_Type;
   protected Mailbox is
      entry Put_Detail(Detail: in Detail_Type);
      entry Get_Detail(Typ: in Integer; Detail : out Detail_Type);
      entry Get_Again(Typ: in Integer; Detail : out Detail_Type);
   private
      Details: Details_Type;
      I, Interested, Details_Count: Integer:=0;
      New_Detail: Boolean := False;
   end Mailbox;
   protected body Mailbox is
      entry Put_Detail(Detail: in Detail_Type)
         when Detail_Count<Capacity is
      begin
         I:=1;
         --Find free slot
         while Details(I).Typ/=0 loop
            I:=I+1;
         end loop;
         Details(I):=Detail;
         Detail_Count:=Detail_Count+1;
         --Number of queuers - may be interested in new detail
         Interested:=Get_Again'Count;
      end Put_Detail;
      entry Get_Detail(Typ: in Integer; Detail : out Detail_Type)
         when True is
      begin
         I:=1;
         while Details(I).Typ/=Typ and I<Capacity loop
            I:=I+1;
         end loop;
         if Details(I).Typ=Typ then
            Detail:=Details(I);
            Details(I).Typ:=0;
            Detail_Count:=Detail_Count-1;
         else
            requeue Get_Again;
         end if;
      end Get_Detail;
      entry Get_Again(Typ: in Integer; Detail : out Detail_Type)
         when Interested>0 is
      begin
         Interested:=Interested-1;
         --Another queuer has had the chance to check mailbox
         I:=1;
         while Details(I).Typ/=Typ and I<Capacity loop
            I:=I+1;
         end loop;
         if Details(I).Typ=Typ then
            Detail:=Details(I);
            Details(I).Typ:=0;
            Detail_Count:=Detail_Count-1;
         else
            requeue Get_Again;
         end if;
      end Get_Again;
   end Mailbox;
```

```
begin
   null;
end Robotsb;
```

c)

```
procedure Robotsc is
   subtype Id_Type is Integer;
   Capacity: Integer:=100;
   Detail_Count: Integer:=0;
   type Detail_Type is record
      Id, Typ: Integer:=0;
   end record;
   type Details_Type is array (1..Capacity) of Detail_Type;
   protected Mailbox is
      entry Put_Detail(Detail: in Detail_Type);
      entry Get_Detail(Typ: in Integer; Detail : out Detail_Type);
      entry Get_Again(Typ: in Integer; Detail : out Detail_Type);
   private
      Details: Details_Type;
      I, Interested, Details_Count: Integer:=0;
      New_Detail: Boolean := False;
   end Mailbox;
   protected body Mailbox is
      entry Put_Detail(Detail: in Detail_Type)
         when Detail_Count<Capacity is
      begin
         I:=1;
         --Find free slot
         while Details(I).Typ/=0 loop
            I:=I+1;
         end loop;
         Details(I):=Detail;
         Detail_Count:=Detail_Count+1;
         --Number of queuer’s - may be interested in new detail
         Interested:=Get_Again'Count;
      end Put_Detail;
      entry Get_Detail(Typ: in Integer; Detail : out Detail_Type)
         when Put_Detail'Count=0 and Details_Count<Capacity is

      begin
         I:=1;
         while Details(I).Typ/=Typ and I<Capacity loop
            I:=I+1;
         end loop;
         if Details(I).Typ=Typ then
            Detail:=Details(I);
            Details(I).Typ:=0;
            Detail_Count:=Detail_Count-1;
         else
            requeue Get_Again;
         end if;
      end Get_Detail;
      entry Get_Again(Typ: in Integer; Detail : out Detail_Type)
         when Interested>0 is
      begin
         Interested:=Interested-1;
         --Another queuer has had the chance to check mailbox
         I:=1;
         while Details(I).Typ/=Typ and I<Capacity loop
            I:=I+1;
         end loop;
         if Details(I).Typ=Typ then
            Detail:=Details(I);
            Details(I).Typ:=0;
            Detail_Count:=Detail_Count-1;
```

```
         else
            requeue Get_Again;
         end if;
      end Get_Again;
   end Mailbox;
begin
   null;
end Robotsc;
```

20.

```
procedure Mailprog is
   type Msg_Type is new Integer;

   task type Mailbox is
      entry Put_Message( Msg : in Msg_Type );
      entry Get_Message( Msg : out Msg_Type);
   end Mailbox;

   task body Mailbox is
      Contents : Msg_Type;
   begin
      loop
         accept Put_Message( Msg: in Msg_Type) do
            Contents := Msg;
         end Put_Message;

         accept Get_Message( Msg: out Msg_Type) do
            Msg := Contents;
         end Get_Message;

      end loop;
   end Mailbox;

   type Mailpek is access Mailbox;

   task Server is
      entry Request(Box: Mailpek; Given_Info: Msg_Type);
   end Server;

   task body Server is
      Reply: Mailpek;
      Info: Msg_Type;
   begin
      loop
         accept Request(Box: Mailpek; Given_Info: Msg_Type) do
            Reply:=Box;
            Info:=Given_Info;
         end Request;
         --process(info);
         Reply.Put_Message(Info);
      end loop;
   end Server;

   task User;
   task body User is
      My_Box: Mailpek;
      My_Info: Msg_Type;
   begin
      Server.Request(My_Box,My_Info);
   end User;

begin
```

```
   null;
end Mailprog;
```

Solutions: Exception handling in Ada95

21.

```
with Ada.Exceptions;  use Ada.Exceptions;

procedure main is
      begin
            NULL;

      exception
        when Constraint_Error => Put ("Exception: Constraint_Error");
        when Numeric_Error =>    Put ("Exception: Numeric_Error");
        when Program_Error =>    Put ("Exception: Program_Error");
        when Storage_Error =>    Put ("Exception: Storage_Error");
        when Tasking_Error =>    Put ("Exception: Tasking_Error");

      end main;
```

22.

```
with Ada.Exceptions;  use Ada.Exceptions;

procedure X is

      App_Exception : exception;

      begin
            NULL;

      exception
        when App_Exception =>  Put ("Procedure X: App_Exception");
        when Event : others => Put ("Procedure X: ");
                               Put ( Exeption_Name( Event ) );
                               Put ( Exeption_Message( Event ) );

      end X;
```

23.

```
...
  exception
     when My_Recoverable_Exception =>
        begin -- attempt recovery
            Recover;
            exception
                  when My_Recoverable_Exception =>
                  Abandon; -- recovery failed!
        end;
```

Solutions: Low level programming in Ada95

24. a)

```
type BYTE is range 0..255; -- Named type and min..max values
for BYTE’SIZE use 8;     -- Object type  needs 8 bits (as character)
```

b)

```
type HIGH_NIBBLE_TYPE  is range 0..15; -- Named type and min..max values
type LOW_NIBBLE_TYPE is range 0..15; -- Named type and min..max values
type NIBBLES  is
      record
            high_nibble: HIGH_NIBBLE_TYPE;
            low_nibble : LOW_NIBBLE_TYPE;
      end record;

      for NIBBLES use
      record
            high_nibble at 0 range 0..3;   -- means b7-b4 in big endian
            low_nibble  at 0 range 4..7;   -- means b3-b0 in big endian
      end record;
```

c)

```
b.high_nibble = a.low_nibble;
b.low_nibble =  a.high_nibble;
```

25.

```
type BYTE is range 0..255;
for BYTE’SIZE use 8;     -- Object type  needs 8 bits (as character)
D_reg: BYTE;
```

Now, force this object to a constant address with use of an *address clause*, e.g.:

```
D_reg_addr: constant System.address := System.Storage_elements.
                                          to_address(16#FFFFFF03#);
for D_reg’address use D_reg_addr;   --  address clause
```

And finally, the function...

```
function ReadRegister return BYTE is
begin
      return(D_reg);
end ReadRegister;
```

26. a)

```
type BIT_TYPE is range 0..1; -- Named type and min..max values
for BIT_TYPE’SIZE use 1;       -- Object type  needs a bit
type DATA_FIELD_TYPE is range 0..63; -- Named type and min..max values
for DATA_FIELD_TYPE’SIZE use 6;      -- Object type  need 6 bits
type PORT is
record
      Rdy:  BIT_TYPE;
      Data: DATA_FIELD_TYPE;
      Ack:  BIT_TYPE;
end record;

for PORT type use
record
-- ADA little endian STILL means bit-swapping…
      Rdy   at 0 range 0..0;  -- means b7 big endian
      Data  at 0 range 1..6;  -- means b6-b1 in big endian
      Ack   at 0 range 7..7;  -- means b0 in big endian
end record;
```

b)

`port_inst:  PORT;` -- instansiation of object PORT_INST of type PORT

-- address clause for this object:

```
for port_inst'address use constant System.address :=
System.Storage_elements.to_address(16#FFFFFF00#);
```

c)

```
function X return BOOLEAN is
begin
      if( port_inst.Rdy)
            X = TRUE;
      X = FALSE;
end;
```

d)

```
function Y return PORT is
begin
      Y = port_inst.Data;
end;
```

e)

```
procedure Z is
begin
      port_inst.Ack = 1;
end;
```

f)

Most students find it difficult to establish correct data type definitions. Once this is done coding is trivial. Most obviously, type declarations go to specification files. Also, functions and procedure declarations go here, maybe to make the functions/procedures visible and at least make the declarations consistent with the corresponding bodies. The difference here is that type, function and procedure declarations in a specification file do NOT produce any code while a body declaration (in a body file) does. Specification files and body files are compiled separately so a difference will generate an error first at link time. The consequense is that you can write all specifications (omitting actual code) and compile these for consistency; you may then start to write the code, adhereing to the specifications, and hopefully end up in a working solution. And as I said in the first sentences, coding is trivial once you have established the specifications.

27.

```
-- nybble.adb

with unchecked_conversion;

package body NYBBLE is

function to_byte is new
      unchecked_conversion( LOW_NIBBLE_TYPE, BYTE );

function to_byte is new
      unchecked_conversion( HIGH_NIBBLE_TYPE, BYTE );

      procedure wnibble ( W : LOW_NIBBLE_TYPE ) is
      begin
            D_reg := to_byte( W );
      end;

      procedure wnibble ( W : HIGH_NIBBLE_TYPE ) is
      begin
            D_reg := to_byte( W );
      end;

end NYBBLE;
```

28.

```
type BYTE is range 0..255;
DATA, STATUS : BYTE;

for DATA'address use constant System.address :=
      System.Storage_elements.to_address(16#FFFFFF03#);
for STATUS'address use constant System.address :=
      System.Storage_elements.to_address(16#FFFFFF05#);

pragma Volatile( STATUS );
pragma Volatile( DATA );

procedure ReadRegister(valid : out BOOLEAN; rdata: out BYTE) is
begin
      if STATUS /= 0
            -- "fresh" data
            valid := TRUE;
      else
            valid := FALSE;
      end if;
      rdata = DATA;

end ReadRegister;
```

29.

```
type BYTE is range 0..255;
DATA  : BYTE;
for DATA'address use constant System.address :=
      System.Storage_elements.to_address(16#FFFFFF03#);
pragma Volatile( DATA );

Dev_priority: constant := implementation defined object priority
Int_Id: Constant := implementation defined hardware priority
```

a)

```
protected Interrupt is
            entry Read(D: out BYTE);
            procedure Handler;
            pragma Interrupt_Priority(Dev_priority);
            pragma Attach_handler(Handler, Int_Id );
      private
            Interrupt_Occured : Boolean := False;
            Buffer: BYTE;
end Interrupt;

protected body Interrupt is

      procedure Handler is
      begin
            Buffer := DATA;
            Interrupt_Occured := True;
      end Handler;

      entry Read(D: out BYTE)
          when Interrupt_Occured is
      begin
            D := Buffer;
            Interrupt_Occured := False;
      end Read;
end Interrupt;
```

b)

```
protected Interrupt is
            entry Read(D: out BYTE);
            procedure Handler;
```

```
            pragma Interrupt_Priority(Dev_priority);
            pragma interrupt_handler(Handler);
        private
            Interrupt_Occured : Boolean := False;
            Buffer: BYTE;
    end Interrupt;

    protected body Interrupt is

        procedure Handler is
        begin
            Buffer := DATA;
            Interrupt_Occured := True;
        end Handler;

        entry Read(D: out BYTE)
           when Interrupt_Occured is
        begin
            D := Buffer;
            Interrupt_Occured := False;
        end Read;
    end Interrupt;

begin
    Attach_Handler(Handler'Access,Int_Id);
end;
```

30.

a)

```
type FLAG is (CLEAR,SET);
for FLAG use (CLEAR => 0, SET => 1); -- Enumeration clause
for FLAG'Size use 1;     -- Size clause

type CHAN_TYPE is range 0..63;
for CHAN_TYPE'Size use 6;      -- "Bit field"  CHAN_TYPE needs 6 bits

type Control_Register is  -- A suitable control register definition
    record
        AD_Start: FLAG;
        Int_Enable: FLAG;
        Done: FLAG;
        Channel: CHAN_TYPE;
        Error: FLAG;
    end record;
```

-- A *record representation clause* is used to define actual bit positions (bit-field position)

```
for Control_Register use
    record
        AD_Start    at 0 range 0..0;
        Int_Enable  at 0 range 6..6;
        Done        at 0 range 7..7;
        Channel     at 0 range 8..13;
        Error       at 0 range 15..15;
    end record;
```

-- Tell compiler the size of this register with a *size clause*:

```
for Control_Register'Size use 16;    -- Type requires 16 bits
                                     -- undefined bits are not used
```

b)

```
with System, System.Storage_elements;
use System;

package body Cntrl_Reg is
    -- Declare the register at FFFFFF04h
    C_reg: Control_Register;
```

```
      for C_reg'address use: constant Address := Storage_elements.
                                      to_address(16#FFFFFF04#);

      function Read_Done return FLAG is
      begin
            return(C_reg.Done);
      end Read_Done;

      function Read_Error return FLAG is
      begin
            return(C_reg.Error);
      end Read_Error;

end Cntrl_Reg;
```

c)

```
procedure Write_Reg(ADS_Flag, I_ED_Flag: in FLAG; Ch: in CHAN_TYPE) is
      Shadow_Register: Control_Register;
begin
      Shadow_Register:= (AD_Start => ADS_Flag,
                         Int_Enable => I_ED_Flag,
                         Done => CLEAR,
                         Channel => Ch,
                         Error => CLEAR);
      C_reg := Shadow_Register;  -- Register update
end Write_Reg;
```

d)

```
-- Package specification *.ads
package AD_Converter is
      Max_Measure : constant := (2**16)-1;  -- 16 bits reg
      subtype MEASUREMENT is Integer range 0..Max_Measure;
      type Chan_Type is range 0..63;      -- 64 channels
      procedure Read_AD(Ch: in CHAN_TYPE; M:
            out MEASUREMENT ; AD_busy:BOOLEAN);
      Conversion_error : exception;
end AD_Converter;

-- Package body *.adb
with System, System.Storage_elements, Ada.Interrupts,
      Ada.Interrupts.Names;
use System, System.Storage_elements, Ada.Interrupts,
      Ada.Interrupts.Names;

package body AD_Converter is
-- type declarations goes here, see (a ...

-- register declarations
      C_reg: Control_Register;
      for C_reg'address use: constant Address :=
            to_address(16#FFFFFF04#);

      D_reg: MEASUREMENT;
      for D_reg'address use constant Address :=
            to_address(16#FFFFFF02#);

      AD_Dev_priority: constant := implementation defined

      protected type AD_Device_Interrupt is
            entry wait_for_completion(M: out MEASUREMENT);
            procedure Handler;
            pragma Interrupt_Priority(AD_Dev_priority);
      pragma Interrupt_handler(Handler);
            Interrupt_Occured : Boolean := False;
      end AD_Device_Interrupt;

      protected body AD_Device_Interrupt is
```

```
        procedure Handler is
        begin -- Interrupt
              Interrupt_Occured := True;
        end Handler;

        entry wait_for_completion (M: out MEASUREMENT)
              when Interrupt_Occured is     -- Wait_for_Interrupt
        begin
              if C_reg.Done = SET and C_reg.Error = CLEAR then
                    -- C_Reg_OK
                    M := D_reg;
              else
                    -- C_Reg_Not_OK
                    interrupt_occured := FALSE;
                    raise Conversion_error;
              end if;
              interrupt_occured := FALSE;
        end wait_for_completion;

  end AD_Device_Interrupt;

  procedure Read_AD(Ch: in CHAN_TYPE; M:
        out MEASUREMENT ; AD_busy:BOOLEAN) is

  begin
        if C_reg.Done = CLEAR
              AD_Busy := TRUE;
        else
              Write_Reg(1, 1; Ch); -- see c)
              wait_for_completion( M );
              AD_Busy := FALSE;
        end if;

  end   Read_AD;


  AD_Interrupt : AD_Device_Interrupt;
  Int_Id: Constant := --implementation defined HW priority;


begin
  Attach_Handler(AD_Interrupt.Handler'Access,Int_Id);
end AD_Converter;
```

31.

```
-- Package specifikation *.ads
package Thermometer is
      procedure Current_Temp(degrees: out integer);
end Thermometer;

with Ada.Interrupts, Ada.Interrupts.Names;
use Ada.Interrupts, Ada.Interrupts.Names;

package body Thermometer is

   Thermometer_Priority: constant := 104; -- Motsvarar HW prioritet 4

   protected Temp_Reader is
      entry Read_Temp(Degrees: out Integer);
      procedure Tempready;
      pragma Interrupt_Handler(Tempready);
      pragma Interrupt_Priority(Thermometer_Priority);
   private
      entry Temp_Arrival(Degrees: out Integer);
```

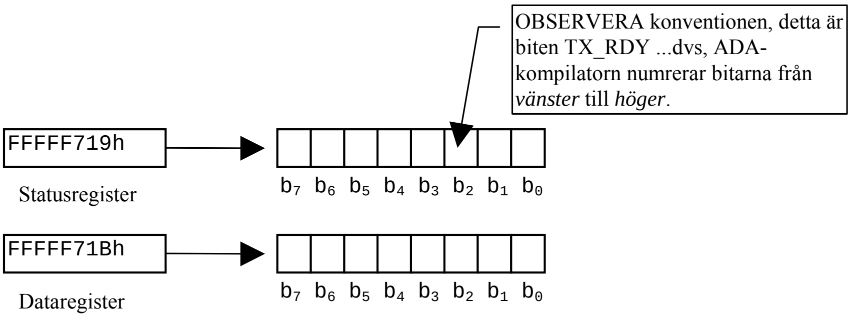
```
      Interrupt_Occured : Boolean := False;
      Next_Request: Boolean := True;
      localDeg: Integer;
   end Temp_Reader;

   protected body Temp_Reader is

      procedure Tempready is
      begin -- Interrupt 4 har skett
         Interrupt_Occured := True;
         Temp_Device(localDeg);
      end Tempready;

      entry Temp_Arrival(Degrees: out Integer)
         when Interrupt_Occured is  -- Wait_On_Interrupt
      begin
         Degrees:=localDeg;
         Next_Request := True;
      end Temp_Arrival; -- Done_Ready

      entry Read_Temp(Degrees: out Integer)
         when Next_Request is -- Wait_On_Done
      begin
         Init_Temp;
         Interrupt_Occured := False;
         Next_Request := False;
         requeue Temp_Arrival;
      end Read_Temp;
   end Temp_Reader;

   procedure Current_Temp(Degrees: out Integer) is
   begin
      Temp_Reader.Read_Temp(Degrees);
   end Current_Temp;

   Int_Id: constant := Ada.Interrupts.Names.Portbint;

begin
   Temp_Reader.Tempready;
   Attach_Handler(Temp_Reader.Tempready'access,Int_Id);
   -- Proceduren TempReady i det skyddade objektet Temp_Reader kopplas
till
   --avbrottet Int_Id
end Thermometer;
```

32.

```
   -- dynamic style
   TimerPriority : constant := implementation defined
   package Timer is
      protected TimerInterface is
            procedure   Handler;
            procedure   InitTimer;
            pragma Interrupt_Priority ( TimerPriority );
            pragma Interrupt_Handler ( Handler );
      end TimerInterface;

      protected body TimerInterface is
            procedure Handler is
            begin
                  -- do nothing right now
            end Handler;

            procedure InitTimer is
            begin
                  -- set up timer device
                  Attach_Handler( Handler, TimerPriority );
```

```
            end InitTimer;

      end TimerInterface;

   end Timer;
```

33.
a)



.ads:

```
with System.Storage_Elements;
package Ex27 is

   type Bit is (Off, On);
   for Bit use (Off=>0, On=>1);

   type Bits is range 0..255;
   for Bits'Size use 8;

   type Sia_Ctrl is
   record
      Tx_Rdy : Bit;     -- transmitter ready
   end record;
   for Sia_Ctrl'Size use 8;    -- 8 bitars register
   for Sia_Ctrl use
   record
      Tx_Rdy at 0 range 5..5; -- Bit 5
   end record;

   -- Register deklarationer och adresser
   D_Sr : Sia_Ctrl;
   for D_Sr'Address use
System.Storage_Elements.To_Address(16#FFFFF719#);
      D_Tb : Bits;
      for D_Tb'Address use
System.Storage_Elements.To_Address(16#FFFFF71B#);

      task Crypto;

      end Ex27;
```

.adb:

```
with Unchecked_Conversion, inbuf;
package body Ex27 is

   function Char_To_Bits is new Unchecked_Conversion(Character,Bits);

   task body Crypto is
      C: Character;
   begin
      while True loop
         Inbuf.Get(C);
         while Sia_Ctrl.Tx_Rdy=Off loop
            --Sia_Ctrl.Tx_Rdy är normalt ON ty seriekretsen snabb
```

```
            null;
         end loop;
         D_Tb:=Char_To_Bits(C);
      end loop;
    end Crypto;
end Ex27;
```

b)
i .ads
enligt exempel 27)

i .adb:

```
with Unchecked_Conversion, inbuf,utbuf;
with Ada.Interrupts.Names;

package body Ex28 is

   function Char_To_Bits is new Unchecked_Conversion(Character,Bits);
   function Bits_To_Char is new Unchecked_Conversion(Bits,Character);

   protected Avbrottshantering is
      pragma Interrupt_Priority (105);
      procedure Avbrott;
      pragma Interrupt_Handler(Avbrott);
      entry Encrypt(C: in Character; Ec: out Character);
   private
      Arrived, Working: Boolean:=False;
      entry Again(C: in Character; Ec: out Character);
   end Avbrottshantering;

   protected body Avbrottshantering is
      procedure Avbrott is
      begin
         Arrived:=True;
      end Avbrott;

      entry Encrypt(C: in Character; Ec: out Character)
         when not Working is
      begin
         while Sia_Ctrl.Tx_Rdy=Off loop
            --Sia_Ctrl.Tx_Rdy är normalt ON ty seriekretsen snabb
            null;
         end loop;
         Working:=True;
         D_Tb:=Char_To_Bits(C);
         requeue Again;
      end Encrypt;

      entry Again(C: in Character; Ec: out Character) when Arrived is
      begin
         Ec:=Bits_To_Char(D_Tb);
         Working:=False;
         Arrived:=False;
      end Again;
   end Avbrottshantering;

   task body Crypto is
      C, Encrypted: Character;
   begin
      while True loop
         Inbuf.Get(C);
         Avbrottshantering.Encrypt(C,Encrypted);
         Utbuf.Put(Encrypted);
```

```
      end loop;
   end Crypto;

ivector: constant := Ada.Interrupts.Names.PortAint;
begin
      Ada.Interrupts.attach_handler(Avbrottshantering.Avbrott'access,
ivector);
end Ex28;
```

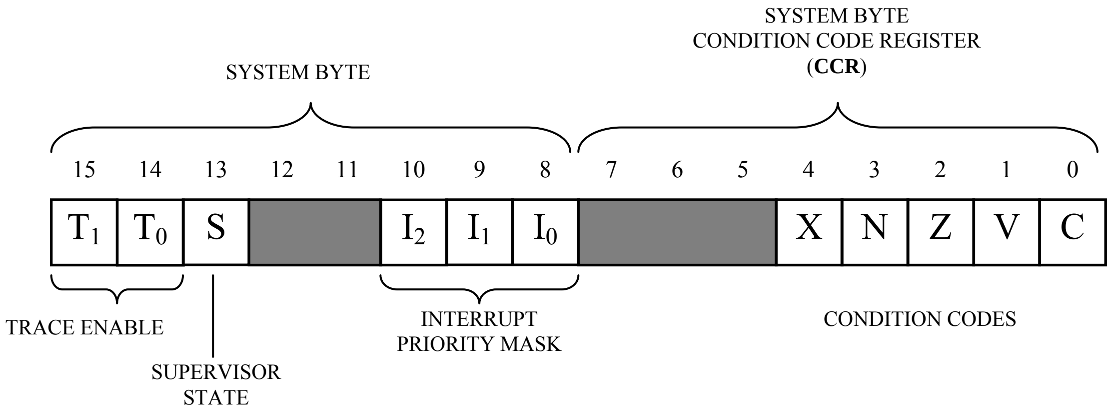
Solutions: Interfacing Ada95 to C and assembly language

34.

```
template_irq:
      movem.l     %A0/%A1/%D0/%D1,-(%SP)  ; save working registers
;     do interrupt handling
      movem.l     (%SP)+, %A0/%A1/%D0/%D1 ; restore working registers
      rte
```

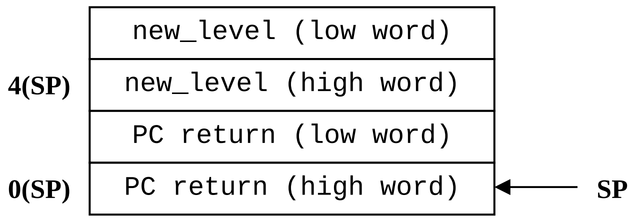
35.

Status register layout:



a)

Call conventions gives Stack contents in “asm_spl”:




```
      .global     asm_spl     ; makes symbol visible globally
asm_spl:
; D0 is the ‘return value’ (by convention)
      clr.l       %D0         ; 0 -> D0
      move        %sr,%D0     ; old SR to D0 low word
      move.l      %D0,%D2     ; a copy to D2
      lsr.l       #8,%D0      ; shift right to correct position
; now set new interrupt priority mask…
      move.l      4(%SP),%D1  ; “new_level” -> D1
      andi.l      #7,%D1      ; make sure ‘new_level’ <= 7
      lsl.l       #8,%D1      ; shift “new_level” to mask position
      andi.l      #0xF8FF,%D2 ; clear mask in old SR copy
      or.l        %D1,%D2     ; D1 = D1 | D2
      move        %D1,sr      ; set new SR
      rts
```

b)

```
      function spl( new_priority_level : in integer )
            return integer;

      pragma Import( c , spl , “asm_spl” );
```

36.

```
      procedure grow( item : in integer; amount : in integer );
      pragma Import( c , grow , “Cgrow” );
```

37.

```
With Interfaces.C; Use Interfaces.C;
…
      function fuzzyval(      item   : in Interfaces.C.int;
                              amount : in Interfaces.C.long )
            return Interfaces.C.int;

      pragma Import( c , fuzzyval , “Cfuzzyval” );
```

38.

a)

```
With Interfaces.C; Use Interfaces.C;
-- package body follows…
      package Ic renames Interfaces.C;    -- create a short name…

      type Struct_Timeval is record
            Tv_Sec      :     Interfaces.C.long;
            Tv_Usec     :     Interfaces.C.long;
      end record;
      pragma Convention( C , Struct_Timeval );
      type Timeval_Ptr is access all Struct_Timeval;

      function Set_Interval_Timer( Timeval_Ptr )
                  return Interfaces.C.int;

      pragma Import( C , Set_Interval_Timer , “SetIntervalTimer”);
```

b)

```
      …
      Itv   :     aliased     Struct_Timeval;
      RetVal:     Interfaces.C.int;
      …
      … assign values to Itv…
      RetVal := Set_Interval_Timer( Itv’access );
```

39.

```
      procedure   Do_Something;
      pragma Export (C, Do_Something, "C_do_something");
      /* the procedure in a C-program */
      extern void do_something (void);
```

Solutions: Worst Case Execution Time estimation

40. Duration (period) of a 100 MHz frequency is 10 ns. Instruction execution time is stated in 'clock cycles' by manufacturers. Every instruction execution time must thus be a multiple of the 10 nanoseconds. Thus 'time unit' = 10 ns is an obvious choice.

41. a) Possible paths are:
    1,2,3,8
    1,2,4,5,8
    1,2,4,6,7,8
    b)
    path: 1,2,3,8 = 4+5+64+1 = 71.
    path: 1,2,4,5,8 = 4+5+5+112+1=127.
    path: 1,2,4,6,7,8=4+5+5+2+112+1=129

42.

Anm: Här kostar deklarationen av F (resp R) lika mycket som deklaration+tilldelning A.

…

Shaws metod ger följande information om programmet (som för övrigt implementerar Fibonaccis välkända algoritm):

Huvudprogrammet:

$WCET(\texttt{Main}) = \{tilldelning, A\} + \{tilldelning, F\} + \{anrop, \texttt{Calculate}\} + WCET(\texttt{Calculate(4)}) + \{tilldelning, F\} = 1 + 1 + 1 + WCET(\texttt{Calculate(4)}) + 1 = 4 + WCET(\texttt{Calculate(4)})$

Funktionen `Calculate`:

$WCET(\texttt{Calculate}(Z)) = \{tilldelning, R\} + \{if\ Z == 0\} + \max(\{tilldelning, R\}, \{if\ Z == 1\} + \max(\{tilldelning, R\}, Dubbelanrop + \{tilldelning, R\})) + \{retursats\}$

där

$Dubbelanrop = \{subtrahera, Z-1\} + \{anrop, \texttt{Calculate}\} + WCET(\texttt{Calculate(Z}-1)) + \{subtrahera, Z-2\} + \{anrop, \texttt{Calculate}\} + WCET(\texttt{Calculate(Z}-2)) + \{addera, \texttt{Calculate(Z}-1) + \texttt{Calculate(Z}-2)\} = 2+1+WCET(\texttt{Calculate(Z}-1))+2+1+WCET(\texttt{Calculate(Z}-2))+2 = 8+WCET(\texttt{Calculate(Z}-1))+ WCET(\texttt{Calculate(Z}-2))$

Eftersom WCET för funktionen `Calculate` är en funktion av sitt eget WCET (två rekursiva anrop) så är det enklast att ta fram de enskilda WCET för givna invärden till funktionen. Då får vi:

$WCET(\texttt{Calculate}(0)) = \{tilldelning, R\} + \{if\ Z == 0\} + \{tilldelning, R\} + \{retursats\} = 1 + 1 + 1 + 1 = 4$

$WCET(\texttt{Calculate}(1)) = \{tilldelning, R\} + \{if\ Z == 0\} + \{if\ Z == 1\} + \{tilldelning, R\} + \{retursats\} = 1 + 1 + 1 + 1 + 1 = 5$

$WCET(\texttt{Calculate}(2)) = \{tilldelning, R\} + \{if\ Z == 0\} + \{if\ Z == 1\} + 8 + WCET(\texttt{Calculate}(1)) + WCET(\texttt{Calculate}(0)) + \{tilldelning, R\})) + \{retursats\} = 1 + 1 + 1 + 8 + 5 + 4 + 1 + 1 = 22$

$WCET(\texttt{Calculate}(3)) = \{tilldelning, R\} + \{if\ Z == 0\} + \{if\ Z == 1\} + 8 + WCET(\texttt{Calculate}(2)) + WCET(\texttt{Calculate}(1)) + \{tilldelning, R\})) + \{retursats\} = 1 + 1 + 1 + 8 + 22 + 5 + 1 + 1 = 40$

$WCET(\texttt{Calculate}(4)) = \{tilldelning, R\} + \{if\ Z == 0\} + \{if\ Z == 1\} + 8 + WCET(\texttt{Calculate}(3)) + WCET(\texttt{Calculate}(2)) + \{tilldelning, R\})) + \{retursats\} = 1 + 1 + 1 + 8 + 40 + 22 + 1 = 74$

WCET för hela programmet blir alltså:

$WCET(\texttt{Main}) = 4 + WCET(\texttt{Calculate}(4)) = 4 + 74 = 78$ mikrosekunder

Detta program är alltså ett bra exempel på hur viktigt det är med full kännedom om indata till vissa typer av program. Utan kännedom om värdet på A i huvudprogrammet går det överhuvudtaget inte att finna ett begränsat värde på WCET för programmet.

.

43.

c) Shaws metod ger följande information om programmet:

Funktionen `Select`:

$WCET(\texttt{Select}(X,Y,Z)) =$
$\{if\ \ Y < Z\} + \max(\{tilldelning, R\}, \{if\ \ X < Z\} + \{tilldelning, R\}) + \{retursats\} =$
$1 + \max(1, 1+1) + 1 = 4$

Kod som omsluts av de nästade looparna (antag 3μs för administration av inre loop):

$WCET(Loopkropp) = \{for\ j\} + \{subtraktion, j-1\} + \{addition, j+1\} + 3 \cdot \{tilldelning, anropsparameter\} +$
$\{anrop, \texttt{Select}\} + WCET(\texttt{Select}()) + \{tilldelning, F\} + \{addition, F + \texttt{Select}()\} +$
$\{tilldelning, C\} + \{addition, C+1\} = 3+2+2+3+1+4+1+2+1+2 = 21$

Funktionen `Calculate` (antag 1μs för administration av yttre loop):

$WCET(\texttt{Calculate}(M)) = \{tilldelning, F\} + \{tilldelning, C\} + 6 \cdot \{for\ i\} +$
$(6+5+4+3+2+1) \cdot WCET(Loopkropp) + \{division, F/C\} + \{retursats\} =$
$1 + 1 + 6 \cdot 1 + 21 \cdot WCET(Loopkropp) + 5 + 1 = 2 + 6 \cdot 1 + 21 \cdot 21 + 6 = 455$

Programkoden för funktionen `Calculate` tar alltså maximalt 455 μs att exekvera. Detta svar får man om man antar att administrationen av den yttre loopen tar 1 μs och för den inre loopen tar 3 μs. Beroende på vilket antagande man gjort angående loopadministrationen kan det erhållna svaret naturligtvis bli både lite större och lite mindre än ovanstående värde. Denna lösning antar dessutom att det kostar 1 μs att hämta ett element från matrisen `M` och använda det som parameter vid anropet till `Select`. Om denna kostnad ignoreras blir programkodens längsta exekveringstid istället 392 μs.

Solutions: Processor utilisation analysis

44. a) It's straight forward that's LCM is 100, a "formal" check yields:

$$\frac{100}{100} = 1, \quad \frac{100}{50} = 2, \frac{100}{25} = 4$$

i.e there is no smaller divisor, so *Least Common Multiple* must be 100.

b) Utilization factor (sum up c/p):

$$\frac{22}{100} + \frac{10}{50} + \frac{8}{25} \approx 0{,}74 = 74\%$$

45. RMSA requires the following condition to be fulfilled:

$$\sum_{i=1}^{n} \frac{c_i}{p_i} \leq n\,(2^{1/n} - 1)$$

Evaluation of left hand yields:

$$\frac{1}{7} + \frac{1}{14} + \frac{4}{18} \approx 0{,}43$$

Evaluation of right hand (n=3) yields:

$$3\,(2^{1/3} - 1) \approx 0{,}78$$

I.e. inequality is true and thereby the task set is schedulable.

Solutions: Response time analysis

46. a) Utilization factor (left hand) for this task set is approx. 0,817.
Calculating right hand yields approx. 0780.
Thus LH > RH and the required condition for schedulability according to RMSA is NOT met.

b) Applying response time analysis:

$$R_i^{n+1} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j$$

Response time for R_a:

$$R_a^0 = 10$$

Response time for R_b:

$$R_b^0 = 10$$

$$R_b^1 = 10 + \left\lceil \frac{10}{30} \right\rceil 10 = 10 + 10 = 20$$

$$R_b^2 = 10 + \left\lceil \frac{20}{30} \right\rceil 10 = 10 + 10 = 20$$

Response time for R_c:

$$R_c^0 = 14$$

$$R_c^1 = 14 + \left\lceil \frac{14}{30} \right\rceil 10 + \left\lceil \frac{14}{40} \right\rceil 10 = 14 + 10 + 10 = 34$$

$$R_c^2 = 14 + \left\lceil \frac{34}{30} \right\rceil 10 + \left\lceil \frac{34}{40} \right\rceil 10 = 14 + 20 + 10 = 44$$

$$R_c^3 = 14 + \left\lceil \frac{44}{30} \right\rceil 10 + \left\lceil \frac{44}{40} \right\rceil 10 = 14 + 20 + 20 = 54$$

$$R_c^4 = 14 + \left\lceil \frac{54}{30} \right\rceil 10 + \left\lceil \frac{54}{40} \right\rceil 10 = 14 + 20 + 20 = 54$$

Gives us:

Task	Period *T* [ms]	Deadline *D* [ms]	Execution time *C* [ms]	***R***
A	30	30	10	10
B	40	40	10	20
C	60	60	14	54

I.e: $R \le D$ for tasks A,B and C, which proves that the task set is in fact schedulable according to response time analysis.

47. a) Order is PA, PB and the response time for PB becomes: 3+4 =7, FAIL.

b) Order is PB, PA and the response time for PA becomes: 4+3 =7, OK.

48. Task execution order is: C,B,A

Response time for R_A	$R_A^{i+1} = \lceil R_A^i/T_B \rceil * C_B + \lceil R_A^i/T_C \rceil * C_C + C_A = \lceil R_A^i/40 \rceil * 10 + \lceil R_A^i/30 \rceil * 10 + C_A$

i=0	0+0+15=15
i = 1	10+10+15=35
i = 2	10+20+15=45
i = 3	20+20+15=**55**
i = 4	20+20+15=**55**

Response time for R_B	$R_B^{i+1} = \lceil R_B^i/T_C \rceil * C_C + C_B = \lceil R_B^i/30 \rceil * 10 + 10$

1	0+10=10
2	10+10 =**20**
3	10+10 =**20**

$R_C = C_C =$ **10**

Check: 55< 65, 20 < 40, 10 < 12, OK.

49.

a) Utilisation:
(3 + 100 + 400 + 5*1000/57 + 1 *1000/37 + 1*1000/7)/1000=0.76

b) Rate monotonic priorities:
F, E, C, D, B, A

c) Deadline monotonic priorities:
F, D, A, E, C, B

d) R_B

$R_B^{i+1} = \lceil R_B^i/T_C \rceil * C_C + \lceil R_B^i/T_E \rceil * C_E + \lceil R_B^i/T_A \rceil * C_A + \lceil R_B^i/T_D \rceil * C_D + \lceil R_B^i/T_F \rceil * C_F + C_B =$ $\lceil R_B^i/50 \rceil * 20 + \lceil R_B^i/33 \rceil * 1 + \lceil R_B^i/1000 \rceil * 3 + \lceil R_B^i/57 \rceil * 5 + \lceil R_B^i/7 \rceil * 1 + 10$

1	0+10=10
2	20+1+3+5+2+10=41
3	20+2+3+5+6+10=46
4	20+2+3+5+7+10=47
5	20+2+3+5+7+10=47

R_C

$R_C^{i+1} = \lceil R_C^i/T_E \rceil * C_E + \lceil R_C^i/T_A \rceil * C_A + \lceil R_C^i/T_D \rceil * C_D + \lceil R_C^i/T_F \rceil * C_F + C_C =$ $\lceil R_C^i/33 \rceil * 1 + \lceil R_C^i/1000 \rceil * 3 + \lceil R_C^i/57 \rceil * 5 + \lceil R_C^i/7 \rceil * 1 + 10$

1	0+20=20
2	1+3+5+3+20=32
3	2+3+5+5+20=35
4	1+3+5+5+20=35

R_E

$R_E^{i+1} = \lceil R_E^i/T_A \rceil * C_A + \lceil R_E^i/T_D \rceil * C_D + \lceil R_E^i/T_F \rceil * C_F + C_E = \lceil R_E^i/1000 \rceil * 3 + \lceil R_E^i/57 \rceil * 5 + \lceil R_E^i/7 \rceil * 1 + 1$

1	0+1=1
2	3+5+1+1=10
3	3+5+2+1=11
4	3+5+2+1=11

R_A

$R_A^{i+1} = \lceil R_A^i/T_D \rceil * C_D + \lceil R_A^i/T_F \rceil * C_F + C_A = \lceil R_A^i/57 \rceil * 5 + \lceil R_A^i/7 \rceil * 1 + 3$

1	0+3=3
2	5+1+3=9
3	5+2+3=10
4	5+2+3=10

R_D

$R_A^{i+1} = \lceil R_A^i/T_F \rceil * C_F + C_A = \lceil R_A^i/7 \rceil * 1 + 3$

1	0+5=5
2	1+5=6
3	1+5=6

$R_F = 1$

e) Ia all $R \leq D$? => Yes, schedulable!

f) Response times for Rate monotonic priority assignments:

$R_x^{i+1} = C_X + \Sigma \lceil R^i/T_j \rceil * C_j$

R_A

$R_A^{i+1} = \lceil R_A^i/T_B \rceil * C_B + ... + \lceil R_A^i/T_F \rceil * C_F + C_A =$ $\lceil R_A^i/100 \rceil * 10 + \lceil R_A^i/57 \rceil * 5 + \lceil R_A^i/50 \rceil * 20 + \lceil R_A^i/33 \rceil * 1 + \lceil R_A^i/7 \rceil * 1 + C_3$

1	0+3=3
2	10+20+5+1+1+3=40
3	10+20+5+2+7+3=47
4	10+20+5+2+7+3=47

R_B

$R_B^{i+1} = \lceil R_A^i/T_C \rceil * C_C + ... + \lceil R_A^i/T_F \rceil * C_F + C_B = \lceil R_B^i/57 \rceil * 5 + \lceil R_B^i/50 \rceil * 20 + \lceil R_B^i/33 \rceil * 1 + \lceil R_B^i/7 \rceil * 1 + 10$

1	0+10=10
2	20+5+1+2+10=38
3	20+5+2+6+10=43
4	20+5+2+7+10=44
5	20+5+2+7+10=44

R_D

$R_D^{i+1} = \lceil R_D^i/T_E \rceil * C_E + \lceil R_D^i/T_F \rceil * C_F + C_D = \lceil R_D^i/50 \rceil * 20 + \lceil R_D^i/33 \rceil * 1 + \lceil R_D^i/7 \rceil * 1 + 5$

1	0+5=5
2	20+1+2+5=28
3	20+1+4+5=30
4	20+1+5+5=31
5	20+1+5+5=31

R_C

$R_C^{i+1} = \lceil R_C^i/T_E \rceil * C_E + \lceil R_C^i/T_F \rceil * C_F + C_C = \lceil R_C^i/33 \rceil * 1 + \lceil R_C^i/7 \rceil * 1 + 20$

1	0+20=20
2	1+3+20=24
3	1+4+20=25
4	1+4+20=25

R_E

$R_E^{i+1} = \lceil R_E^i/T_F \rceil * C_F + C_E = \lceil R_E^i/7 \rceil * 1 + 1$

1	0+1=1
2	1+1=2
3	1+1=2

R_A 47
R_B 44
R_C 25
R_D 31
R_E 2
R_F 1

g) $R_A=47 > D_A=20$ i.e. FAIL!
$R_D=31 > D_D=10$ ie. FAIL!

h) Response times Deadline monotonic priority assignments

New task set:

	T	D	C
a	1000	20	3
b	100	100	10
c	50	50	20
d	57	10	5
e	33	33	1
f	7	7	1
ft	30	5	2

$R_x^{i+1} = C_X + \Sigma \lceil R^i/T_j \rceil * C_j$

R_B
(start with 47)

$R_B^{i+1} = \lceil R_B^i/T_C \rceil * C_C + \lceil R_B^i/T_E \rceil * C_E + \lceil R_B^i/T_A \rceil * C_A + \lceil R_B^i/T_D \rceil * C_D + \lceil R_B^i/T_F \rceil * C_F + \lceil R_B^i/T_{FT} \rceil * C_{FT} + C_B = \lceil R_B^i/50 \rceil * 20 + \lceil R_B^i/33 \rceil * 1 + \lceil R_B^i/1000 \rceil * 3 + \lceil R_B^i/57 \rceil * 5 + \lceil R_B^i/7 \rceil * 1 + \lceil R_B^i/30 \rceil * 2 + 10$

1	20+2+3+5+7+4+10=51
2	40+2+3+5+8+4+10=72
3	40+3+3+10+11+6+10=83
4	40+3+3+10+12+6+10=84
5	40+3+3+10+12+6+10=84

R_C
(start with 35)

$R_C^{i+1} = \lceil R_C^i/T_E \rceil * C_E + \lceil R_C^i/T_A \rceil * C_A + \lceil R_C^i/T_D \rceil * C_D + \lceil R_C^i/T_F \rceil * C_F + \lceil R_C^i/T_{FT} \rceil * C_{FT} + C_C = \lceil R_C^i/33 \rceil * 1 + \lceil R_C^i/1000 \rceil * 3 + \lceil R_C^i/57 \rceil * 5 + \lceil R_C^i/7 \rceil * 1 + \lceil R_C^i/30 \rceil * 2 + 10$

1	39
2	40
3	40

R_E
(start with 11)

$R_E^{i+1} = \lceil R_E^i/T_A \rceil * C_A + \lceil R_E^i/T_D \rceil * C_D + \lceil R_E^i/T_F \rceil * C_F + \lceil R_E^i/T_{FT} \rceil * C_{FT} + C_E = \lceil R_E^i/1000 \rceil * 3 + \lceil R_E^i/57 \rceil * 5 + \lceil R_E^i/7 \rceil * 1 + \lceil R_E^i/30 \rceil * 2 + 1$

1	13
2	13

R_A
(start with 10)

$R_A^{i+1} = \lceil R_A^i/T_D \rceil * C_D + \lceil R_A^i/T_F \rceil * C_F + \lceil R_A^i/T_{FT} \rceil * C_{FT} + C_A = \lceil R_A^i/57 \rceil *5 + \lceil R_A^i/7 \rceil *1 + \lceil R_A^i/30 \rceil *2+3$

1	12
2	12

R_D
(start with 6)

$R_D^{i+1} = \lceil R_D^i/T_F \rceil * C_F + \lceil R_D^i/T_F \rceil * C_F + C_D = \lceil R_D^i/7 \rceil *1 + \lceil R_{FT}^i/30 \rceil *2+3$

1	8
2	9
3	9

R_F
(start with 1)

$R_F^{i+1} = \lceil R_F^i/T_F \rceil * C_F + \lceil R_F^i/T_F \rceil * C_F + C_F = \lceil R_{FT}^i/30 \rceil *2+3$

1	3
2	3

$R_{FT} = 2$

Task	R	D	
FT	2	5	OK
F	3	7	OK
D	9	10	OK
A	12	20	OK
E	13	33	OK
C	40	50	OK
B	84	100	OK

50.

Med ledning av hur processerna använder semaforerna bestämmer vi prioritetstaken för semaforerna:

uses(P1) = s_1	*ceil*(s_1) = *pri(P1)* = 1
uses(P2) = s_2,s_3	*ceil*(s_2) = *max{ pri(P2),pri(P3)}* =2
uses(P3) = s_3, s_2	*ceil*(s_3) = *max{ pri(P2),pri(P3)}* =2

Vi kan nu bestämma b_{P1}:

- "Undersök alla processer med lägre prioritet", dvs P2 och P3
- "Bestäm vilka semaforer dessa processer använder", dvs s_2 och s_3.
- "Plocka ut de semaforer som har högre (eller samma) prioritetstak som aktuell process", dvs semafortak högre (eller samma) som process P1, några sådana semaforer finns inte här...

Följaktligen blir $b_{P1} = 0$.

Vi bestämmer nu b_{P2}:

- processer med lägre prioritet än P2, dvs P3
- vilka semaforer dessa processer använder", dvs s_2 och s_3.
- semaforer som har högre (eller samma) prioritetstak som P2 av dessa, dvs s_2 och s_3.

b_{P2}=längsta kritiska region, dvs max{$cs_{P3,S2}$, $cs_{P3,S3}$}=25 ms.

Slutligen bestämmer vi b_{P3}:

Det finns inga lägre prioriterade processer varför $b_{P3} = 0$.

Resultaten kan sammanfattas i följande tabell.

Task	*pri*	*T*	*D*	*C*	*B*
A	1	50	10	5	0
B	2	500	500	240	25
C	3	3000	3000	1000	0

Därefter kan analys utföras, på samma sätt som tidigare men med hänsyn tagen till användning av semaforer.

Task	*pri*	*T*	*D*	*C*	*B*	*R*
A	1	50	10	5	0	5
B	2	500	500	240	25	295
C	3	3000	3000	1000	0	2445

I.e. all $R_i < D_i$, schedulable.

51.

$$R_i^{n+1} = C_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j$$

We first have to determine each blocking factor B_i. To do this we need the ceiling priorities.
Let 1 be high priority, then we have P_A (pri) = 1, P_B (pri) = 2, P_C (pri) = 3, P_D (pri) = 4.
Ceiling priorities: For each semaphore, find the highest priority among the tasks that uses this semaphore, this is the semaphores ceiling priority:

ceil{S_1} = max {P_A (pri), P_B (pri)} = max {1,2} = 1
ceil{S_2} = max {P_B (pri), P_C (pri)} = max {2,3} = 2
ceil{S_3} = max {P_A (pri), P_D (pri)} = max {1,4} = 1

Now we identify for each task P_i, which lower priority tasks that may interfere (block P_i).

P_A can be blocked by P_B and P_D since they use semaphores with a ceiling priority that is higher or equal to the priority of P_A.
Blocking factor for P_A becomes:

B_A = max { P_B uses S_1, P_B uses S_2, P_D uses S_3 } =
max { $P_B(H_{S1})$,$P_B(H_{S2})$, $P_D(H_{S3})$ } =
max { 1 , 2 , 2 } = **2**

P_B can be blocked by P_C and P_D.
Blocking factor for P_B becomes:

B_B = max { P_C uses S_2, P_D uses S_3 } =
max { $P_C(H_{S2})$, $P_D(H_{S3})$ } =
max { 3 , 2 } = **3**

P_C can be blocked by P_D.
Blocking factor for P_C becomes:

B_C = max { P_D uses S_3 } =
max { $P_D(H_{S3})$ } =
max { 2 } = **2**

Now, P_D cannot be blocked since there are no lower priority tasks, thus we conclude:

B_A = max{1,2} = 2
B_B = max{3,2} = 3
B_C = max{2}= 2
B_D= 0

Finally we calculate the response times and check all deadlines:

Response time for P_A.

$R_A = C_A + B_A = 2 + 2 = 4 \le D_A = 4$: *OK!*

Response time for P_B.

Iteration	$R_B^{i+1} = C_B + B_B + \lceil R_B^i / T_A \rceil * C_A = C_B + B_B + \lceil R_B^i / 5 \rceil * 2$

Setting start value $R_B^0 = C_B = 3$ yuilds:

1	3 + 3 + 2 = 8
2	3 + 3 + 4 = 10
3	$3 + 3 + 4 = 10 \le D_B = 12$: *OK!*

Response time for P_C.

Iteration	$R_C^{i+1} = C_C + B_C + \lceil R_C^i / T_B \rceil * C_B + \lceil R_C^i / T_A \rceil * C_A =$ $C_C + B_C + \lceil R_C^i / 16 \rceil * 3 + \lceil R_C^i / 5 \rceil * 2$

Setting start value $R_C^0 = C_C = 3$ yuilds:

1	3 + 2 + 3 + 2 =10
2	3 + 2 + 3 + 4 = 12
3	3 + 2 + 3 + 6 = 14
4	$3 + 2 + 3 + 6 = 14 \le D_C = 16$: *OK!*

Response time for P_D.

Iteration	$R_C^{i+1} = C_D + \lceil R_D^i / T_C \rceil * C_C + \lceil R_D^i / T_B \rceil * C_B + \lceil R_D^i / T_A \rceil * C_A =$ $C_D + \lceil R_D^i / 20 \rceil * 3 + \lceil R_B^i / 16 \rceil * 3 + \lceil R_B^i / 5 \rceil * 2$

Setting start value $R_D^0 = C_D = 4$ yuilds:

1	4 + 3 + 3 + 2 =12
2	4 + 3 + 3 + 6 = 16
3	4 + 3 + 3 + 8 = 18
4	4 + 3 + 6 + 8 = 21
5	4 + 6 + 6 + 10 = 26
6	4 + 6 + 6 + 12 = 28
7	$4 + 6 + 6 + 12 = 28 \le D_D = 28$: *OK!*

I.e. All scheduled tasks will meet their deadline.

52. a) RMSA *requires* the following condition to be fulfilled:

$$\sum_{i=1}^{n} \frac{c_i}{p_i} \le n\,(2^{1/n} - 1)$$

Evaluation of left hand yields:

$$\frac{2}{7} + \frac{1}{8} + \frac{3}{10} + \frac{2}{25} \approx 0{,}79$$

Evaluation of right hand (n=4) yields:

$$4(2^{1/4} - 1) \approx 0{,}76$$

I.e. LH > RH, *not* schedulable according to this condition, QED...

b), c)

Task	***T***	***D***	***C***	***pri***	***B***	***R(b)***	***R(c)***
A	7	7	2	0	0	2	2
B	8	8	1	1	1	3	4
C	10	10	3	2	2	6	10
D	25	25	2	3	0	14	14

53.

Standard CAN: $\left\lfloor \frac{34 + 8i}{4} \right\rfloor + 47 + 8i$, where i is the number of data bytes in the message

Extended CAN: $\left\lfloor \frac{54 + 8i}{4} \right\rfloor + 67 + 8i$, where i is the number of data bytes in the message

a) 75
b) 135
c) 110
d) 150

54.
a)

Message	*D(us)*	*T(us)*	*bits*	*C(us)*	*B(us)*	*R(us)*
M1	4000	10000	65	1300	2300	3600
M2	6000	20000	105	2100	2300	5700
M3	10000	15000	115	2300	0	5700

$$R_i^{n+1} = C_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j$$

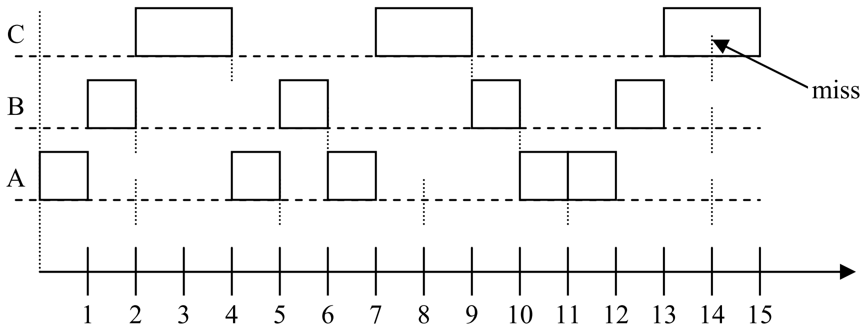
b) Bus utilisation ≈ 39%

Solutions: Processor demand analysis

55. a) Utilization factor:

$$U = \sum_{i=1}^{i=n} \left(\frac{C_i}{T_i} \right) = \frac{1}{3} + \frac{1}{4} + \frac{2}{5} = \frac{20}{60} + \frac{15}{60} + \frac{24}{60} = \frac{59}{60}$$

b) Timing diagram:



c) Processor demand calculation.
LCM { T_A , T_B , T_C } = LCM { 3, 4, 5 } = 60

Determine checkpoints K within interval [0,60].

For T_A we obtain the following checkpoints
$K_A = \{ D^k_A = k*T_A + D_A$, k = 0, 1, 2, 3, ...} = { 2, 5, 8, 11, 14, ...}

For T_B we obtain the following checkpoints
$K_B = \{ D^k_B = k*T_B + D_B$, k = 0, 1, 2, ...} = { 2, 6, 10, 14, ...}

For T_C we obtain the following checkpoints
$K_C = \{ D^k_C = k*T_C + D_C$, k = 0, 1, 2, ...} = { 4, 9, 14, 19, ...}

Processor demand $C_P(0, L)$ thus must be checked for the following checkpoints.
$\boldsymbol{K} = K_A \cup K_B \cup K_C$ = { 2, 4, 5, 6, 8, 9, 10, 11, 14, 19, ...}

Consider the general expression for processor demand analysis:

$$C_p(0,L) = \sum N_i^L \times C_i = \sum \left(\left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right) \times C_i$$

Each checkpoint is analysed in the following table:

L	$N^L{}_A*C_A$	$N^L{}_B*C_B$	$N^L{}_C*C_C$	$C_P(0, L)$	$C_P(0, L) \leq L$
2	(⌊(2-2) / 3⌋)+1)*1 = 1	(⌊(2-2) / 4⌋)+1)*1 = 1	(⌊(2-4) / 5⌋)+1)*2 = 0	2	OK
4	(⌊(4-2) / 3⌋)+1)*1 = 1	(⌊(4-2) / 4⌋)+1)*1 = 1	(⌊(4-4) / 5⌋)+1)*2 = 2	4	OK
5	(⌊(5-2) / 3⌋)+1)*1 = 2	(⌊(5-2) / 4⌋)+1)*1 = 1	(⌊(5-4) / 5⌋)+1)*2 = 2	5	OK
6	(⌊(6-2) / 3⌋)+1)*1 = 2	(⌊(6-2) / 4⌋)+1)*1 = 2	(⌊(6-4) / 5⌋)+1)*2 = 2	6	OK
8	(⌊(8-2) / 3⌋)+1)*1 = 3	(⌊(8-2) / 4⌋)+1)*1 = 2	(⌊(8-4) / 5⌋)+1)*2 = 2	7	OK
9	(⌊(9-2) / 3⌋)+1)*1 = 3	(⌊(9-2) / 4⌋)+1)*1 = 2	(⌊(9-4) / 5⌋)+1)*2 = 4	9	OK
10	(⌊(10-2) / 3⌋)+1)*1 = 3	(⌊(10-2) / 4⌋)+1)*1 = 3	(⌊(10-4) / 5⌋)+1)*2 = 4	10	OK
11	(⌊(11-2) / 3⌋)+1)*1 = 4	(⌊(11-2) / 4⌋)+1)*1 = 3	(⌊(11-4) / 5⌋)+1)*2 = 4	11	OK
14	**(⌊(14-2) / 3⌋)+1)*1 = 5**	**(⌊(14-2) / 4⌋)+1)*1 = 4**	**(⌊(14-4) / 5⌋)+1)*2 = 6**	**15**	**NOT OK!**

I.e. NOT schedulable since $C_P(0, 14) = 15$ exceeds length of the interval.

56.

a) Since $D_i < T_i$ for *at least* on task in the set, the simple L&L test don't apply.

b)

LCM{A,B,C} = LCM{4,10,20} = 20.

Checkpoints ***K***:

K_A = {4,8,12,16,20}

K_B = {4,14}

K_C = {16}

Now: ***K*** = $K_A \cup K_B \cup K_C$ = { 4,8,12,14,16,20}

L	$N^L{}_A$*C_A	$N^L{}_B$*C_B	$N^L{}_C$*C_C	C_P (0, L)	C_P (0, L) ≤ L
4	(⌊(4-4) / 4⌋)+1)*3 = 3	(⌊(4-4) / 10⌋)+1)*1 = 1	(⌊(4-16) / 20⌋)+1)*3 = 0	4	OK
8	(⌊(8-4) / 4⌋)+1)*3 = 6	(⌊(8-4) / 10⌋)+1)*1 = 1	(⌊(8-16) / 20⌋)+1)*3 = 0	7	OK
12	(⌊(12-4) / 4⌋)+1)*3 = 9	(⌊(12-4) / 10⌋)+1)*1 = 1	(⌊(12-16) / 20⌋)+1)*3 = 0	10	OK
14	(⌊(14-4) / 4⌋)+1)*3 = 9	(⌊(14-4) / 10⌋)+1)*1 = 2	(⌊(14-16) / 20⌋)+1)*3 = 0	11	OK
16	**(⌊(16-4) / 4⌋)+1)*3 = 12**	**(⌊(16-4) / 10⌋)+1)*1 = 2**	**(⌊(16-16) / 20⌋)+1)*3 = 3**	**17**	**NOT OK!**
20	(⌊(20-4) / 4⌋)+1)*3 = 15	(⌊(20-4) / 10⌋)+1)*1 = 2	(⌊(20-16) / 20⌋)+1)*3 = 3	20	OK

I.e. NOT schedulable since C_P (0, 16) = 17 exceeds length of the interval.

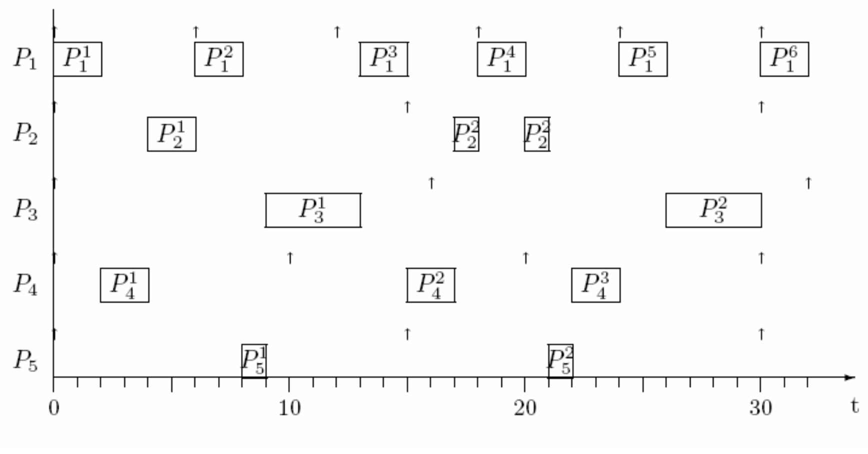
57.

a) Utnyttjandegraden i systemet är:

$$U = \sum_{\forall i} C_i/T_i = 2/6 + 2/15 + 4/16 + 2/10 + 1/15 = 59/60$$

Eftersom $U = 59/60 \leq 1$ och $D_i = T_i$ för alla processer så är processerna schemaläggningsbara med avseende på EDF.

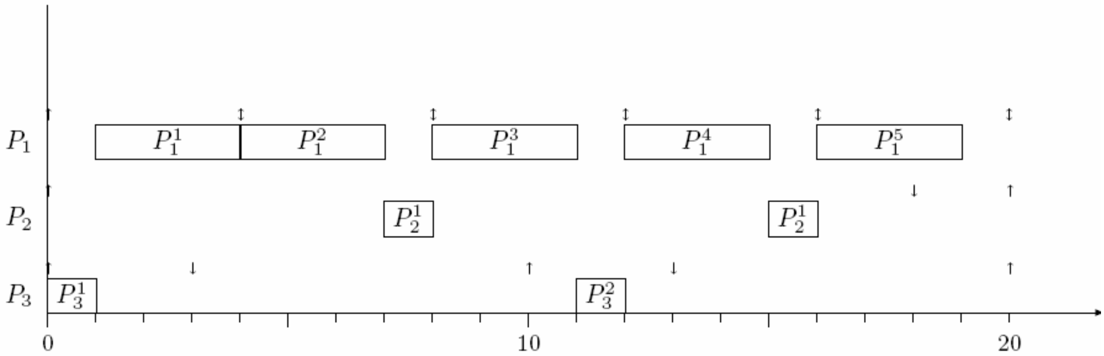
b) En simulering av processerna med EDF schemaläggning ger följande tidsdiagram.



58.

a) Vi räknar först ut LCM för processerna: LCM$\{P_1, P_2, P_3\}$ = LCM$\{4, 10, 20\} = 20$.

En simulering av processerna med *svarstidskritisk* (EDF) schemaläggning under tiden $[0, LCM]$ ger följande tidsdiagram.



b) Tillämpa *beräkningsbehovsanalys* ("processor-demand analysis"):

Vi räknar återigen ut LCM för processerna: LCM$\{P_1, P_2, P_3\}$ = LCM$\{4, 10, 20\} = 20$.

Därefter tar vi (se diagram i deluppgift a) fram kontrollpunktsmängden K: $K_1 = \{4, 8, 12, 16, 20\}$, $K_2 = \{18\}$ och $K_3 = \{3, 13\}$ vilket ger $K = K_1 \cup K_2 \cup K_3 = \{3, 4, 8, 12, 13, 16, 18, 20\}$.

Schemaläggningsanalysen ger nu:

L	$N_1^L \cdot C_1$	$N_2^L \cdot C_2$	$N_3^L \cdot C_3$	$C_P(0, L)$	$C_P(0, L) \leq L$
3	$(\lfloor\frac{(3-4)}{4}\rfloor + 1) \cdot 3 = 0$	$(\lfloor\frac{(3-18)}{20}\rfloor + 1) \cdot 2 = 0$	$(\lfloor\frac{(3-3)}{10}\rfloor + 1) \cdot 1 = 1$	1	OK
4	$(\lfloor\frac{(4-4)}{4}\rfloor + 1) \cdot 3 = 3$	$(\lfloor\frac{(4-18)}{20}\rfloor + 1) \cdot 2 = 0$	$(\lfloor\frac{(4-3)}{10}\rfloor + 1) \cdot 1 = 1$	4	OK
8	$(\lfloor\frac{(8-4)}{4}\rfloor + 1) \cdot 3 = 6$	$(\lfloor\frac{(8-18)}{20}\rfloor + 1) \cdot 2 = 0$	$(\lfloor\frac{(8-3)}{10}\rfloor + 1) \cdot 1 = 1$	7	OK
12	$(\lfloor\frac{(12-4)}{4}\rfloor + 1) \cdot 3 = 9$	$(\lfloor\frac{(12-18)}{20}\rfloor + 1) \cdot 2 = 0$	$(\lfloor\frac{(12-3)}{10}\rfloor + 1) \cdot 1 = 1$	10	OK
13	$(\lfloor\frac{(13-4)}{4}\rfloor + 1) \cdot 3 = 9$	$(\lfloor\frac{(13-18)}{20}\rfloor + 1) \cdot 2 = 0$	$(\lfloor\frac{(13-3)}{10}\rfloor + 1) \cdot 1 = 2$	11	OK
16	$(\lfloor\frac{(16-4)}{4}\rfloor + 1) \cdot 3 = 12$	$(\lfloor\frac{(16-18)}{20}\rfloor + 1) \cdot 2 = 0$	$(\lfloor\frac{(16-3)}{10}\rfloor + 1) \cdot 1 = 2$	14	OK
18	$(\lfloor\frac{(18-4)}{4}\rfloor + 1) \cdot 3 = 12$	$(\lfloor\frac{(18-18)}{20}\rfloor + 1) \cdot 2 = 2$	$(\lfloor\frac{(18-3)}{10}\rfloor + 1) \cdot 1 = 2$	16	OK
20	$(\lfloor\frac{(20-4)}{4}\rfloor + 1) \cdot 3 = 15$	$(\lfloor\frac{(20-18)}{20}\rfloor + 1) \cdot 2 = 2$	$(\lfloor\frac{(20-3)}{10}\rfloor + 1) \cdot 1 = 2$	19	OK

Beräkningsbehovet i varje strategiskt tidsintervall överstiger aldrig intervallets längd så alla processer möter sina deadlines.