Outline

- Interfacing Ada95 to C and assembly language
- Problems demonstrated during exercise: 34,35,38
- WCET analysis
- Problems 40, 42

Ada95, annex B3 “Interfacing with C”

- The facilities relevant to interfacing with the C language are the package `Interfaces.C` and its children; support for the `Import`, `Export`, and `Convention` pragmas.
- The package `Interfaces.C` contains the basic types, constants and subprograms that allow an Ada program to pass scalars and strings to C functions.

Interfacing with C - example

```
-- Calling the C Library Function strcpy
with Interfaces.C;
procedure Test is
   package C renames Interfaces.C;
   use type C.char_array;

   -- Call <string.h>strcpy:
   -- C definition of strcpy:  char *strcpy(char *s1, const char *s2);
   -- Note: since the C function's return value is ignored, the Ada interface is a procedure

   procedure Strcpy (Target : out C.char_array; Source : in  C.char_array);

   pragma Import(C, Strcpy, "strcpy");

   Chars1 :  C.char_array(1..20);
   Chars2 :  C.char_array(1..20);

begin
   Chars2(1..6) := "qwert" & C.nul;
   Strcpy(Chars1, Chars2);
   -- Now Chars1(1..6) = "qwert" & C.Nul
end Test;
```

Interfacing with assembly language

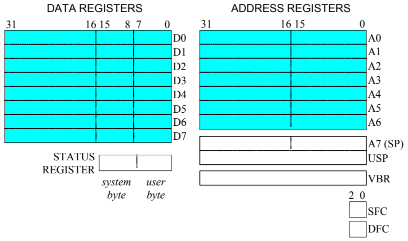
- When writing assembly programs we use the C-compiler conventions for register usage, stack use etc..
- Then, interfacing assembly programs is identical to interfacing C...

Assignment 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.

CPU32 – Programmers model

“working registers”



```
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
```

Assignment 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.

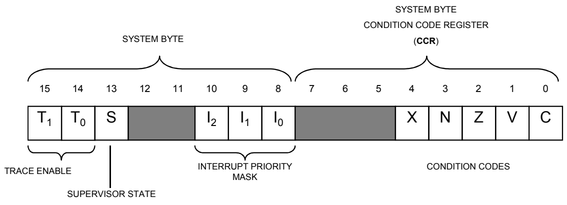
b) Show how to import `asm_spl` to a function `spl` in an Ada-program.

EXAMPLE USE:

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

Solution

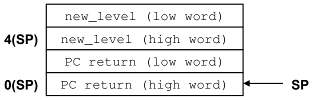
Status register layout:



1. Read IPM, shall be returned as an integer (0..7)
2. Set IPM bits from parameter (integer)

Solution

C-compiler call conventions gives Stack contents in “`asm_spl`”:



```
         .global  asm_spl           ; makes symbol visible globally
asm_spl:
; D0 is the ‘return value’ (by C compiler 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

-- Declarations in Ada95
         function spl( new_priority_level : in integer )
                  return integer;
         pragma Import( c , spl , “asm_spl” );
```

Assignment 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.

Solution 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        :      Ic.long;
      Tv_Usec       :      Ic.long;
   end record;

   pragma Convention( C , Struct_Timeval );
            -- Ada will represent Struct_Timeval as a C-style structure.

   type Timeval_Ptr is access all Struct_Timeval;

   function Set_Interval_Timer( Timeval_Ptr ) return Ic.int;

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

Solution b)

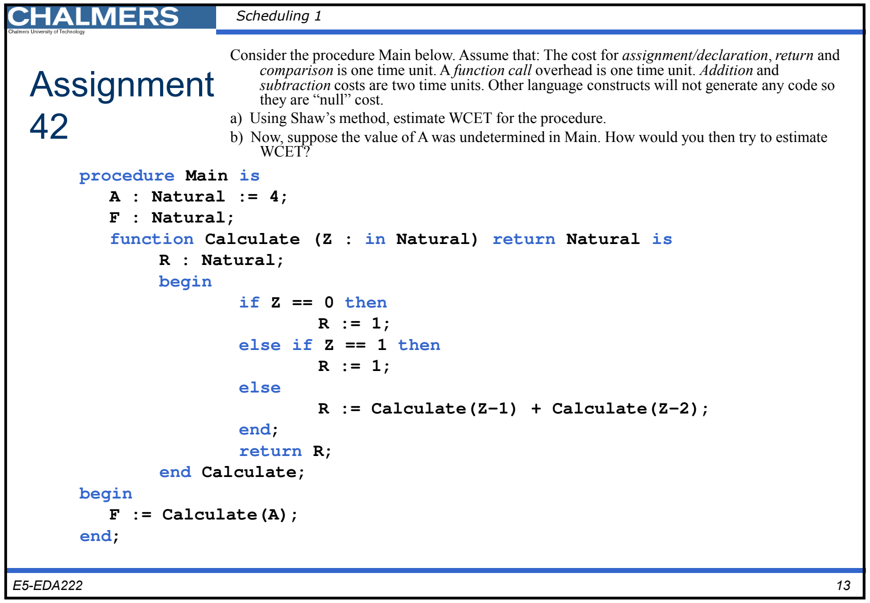
```
Itv :      aliased      Struct_Timeval;
RetVal:    Ic.int;
...
... assign values to Itv…
...
RetVal := Set_Interval_Timer( Itv’access );
```

Assignment 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.

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.



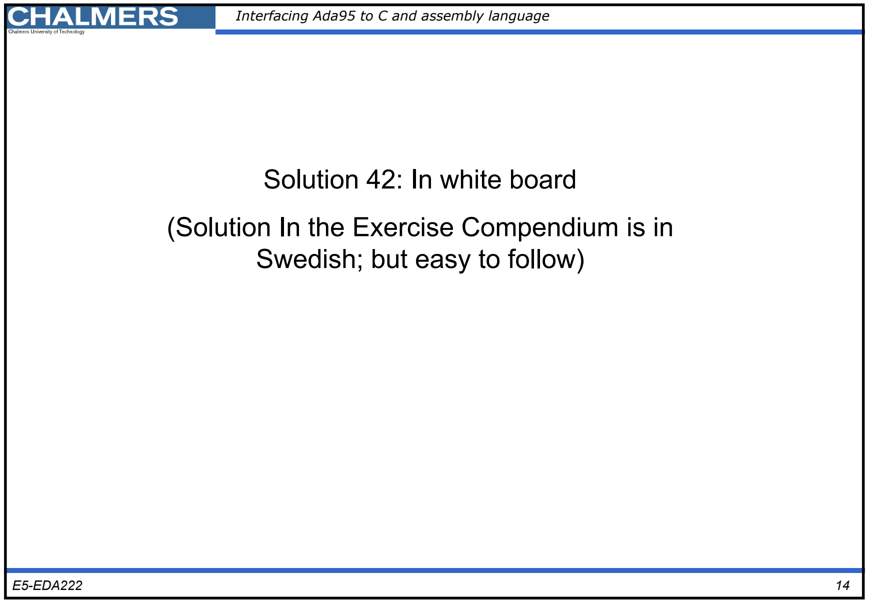
Assignment 42

Consider the procedure Main below. Assume that: The cost for *assignment/declaration*, *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;
```



Solution 42: In white board

(Solution In the Exercise Compendium is in Swedish; but easy to follow)