Oefeningen Computersystemen 14.11.1996

1. Register gebaseerde processorarchitectuur

Beschouw de "RB1" machine. Voor de "opcode" gebruiken we 8 bits. Addressen (indirect en direct) en constanten ("Immediate" addressering) stellen we voor met 32 bit. Het geheugen wordt *per byte* geaddresseerd. Definieer als machine code (voorgesteld in hexadecimale notatie) voor de respectievelijke instructies :

NULL 00 *doet niets, behalve P := P + 1 ;*

Data Manipulatie :

EX	01 R1 R2 *met Ri = 0..7 voor D0..D7 en 8..15 voor A0..A7.*
CS	2 R1
ADD	03 R1 R2 met als resultaat *R1 := R1 + R2*
MIN	04 R1 R2
MUL	05 R1 R2
DIV	06 R1 R2
CMP	7x R1 R2 met x = 0 : EQ?; x = 1 : NE?; x = 2 : LT?; x = 3 : LE?; x= 4 : GT?; x = 5 : GE?; x = 6 : OV?

Data Handling

MOVE R1 OP2	08 M R1 OP2 *van R1 naar OP2*
MOVE OP1 R2	09 M R2 OP1 *van OP1 naar R2*

met Ri : cf. EX

en OPi : de operand volgens de addresseringsmode gecodeerd in M :

- *M = 0 : register, i.e. OPi = 0..15 volgens de codering in EX*
- *M = 1 : condition code register (cc)*
- *M = 2 : immediate 8 bits*
- *M = 3 : immediate 16 bits*
- *M = 4 : immediate 32 bits. In assembler stellen we dit voor als #xxxxxxxx.*
- *M = 5 : direct, i.e. het OPi veld is een 32bit adres. In assembler stellen we dit voor als xxxxxxxx.*
- *M = 6 : indirect register, i.e. OPi = 0.. 7, en aangezien enkel de adresregisters addressen kunnen bevatten is 0 de codering voor A0, 1 voor A1 etc. In assembler stellen we dit voor als (Ai).*

M = 7 : indirect register with post increment : idem behalve dat ***na*** *de transfer Ai verhoogd wordt met 4. In assembler stellen we dit voor als (Ai)+*

M = 8 : indirect register with predecrement : idem als M=6, behalve dat ***voor*** *de transfer Ai verlaagd wordt met 4. In assembler stellen we dit voor als -(Ai)*

M = 9 : indirect memory, i.e. OPi is een 32 bit adres van de geheugenplaats waar het adres van de operand staat. In assembler stellen we dit voor als (xxxxxxxx)

M = A : indirect memory with postincrement. In assembler stellen we dit voor door (xxxxxxxx)+.

M = B : indirect memory with predecrement. In assembler stellen we dit voor door -(xxxxxxxx).

M = C : indexed : OPi bestaat uit een 3 bits veld dat een adres register aanduidt en een offset die bij deze waarde geteld wordt. In assembler stellen we dit voor door (Ai,xxxxxxxx)

M = D : indexed with post increment. In assembler stellen we dit voor door (Ai,xxxxxxxx)+.

M = E : indexed with predecrement. In assembler stellen we dit voor door -(Ai,xxxxxxxx).

<u>Controle Instructies</u>

JMP xxxxxxxx	0A xxxxxxxx *waarbij xxxxxxxx het bestemmingsadres is*
JPC c xxxxxxxx	B c xxxxxxxx *met c = 0 : EQ?; c = 1 : NE? etc (cf. p 5-7)*
JSR xxxxxxxx	0C xxxxxxxx *idem als JMP*
RET	0D

De stack pointer is A5. Deze wijst naar een stuk centraal geheugen dat we als “stack” gebruiken. Als base pointer kiezen we A4. Aangezien bij een call enkel het return adres (i.e. het adres van de instruktie die volgt op JSR xxxxxxxx) op de stapel geplaatst wordt moeten de static en de dynamic link expleciet op de stapel gezet worden [met MOV Ri,(A5)+ instructies]. Het omgekeerde moet natuurlijk gebeuren net voor de RET instructie [met MOV -(A5),Ri instructies].

Ons “stack frame” zal er bij een procedureoproep bijgevolg als volgt uitzien :

	…
A5 (SP) =======→	Local Variable 2
	Local Variable 1
	Dynamic Link
	Static Link
A4 (BP) =======→	Return Address
	Actual Parameter 2
	Actual Parameter 1
	Return Value (voor een functie)

…

Schrijf een assembler programma (i.e. in mnemonics, dus **niet** in machinetaal) dat een Fibonacci getal berekent :

$$fib(n) = fib(n-1) + fib(n-2)$$
$$\text{met } fib(0) = fib(1) = 1.$$

Doe dit op 3 verschillende manieren.

SCHRIJF COMMENTAAR BIJ UW ASSEMBLER PROGRAMMA !!

<u>1.a. Functie en locale variabelen</u>

```
MODULE Fib
PROCEDURE Fibb(n:CARDINAL):CARDINAL;
    VAR tmp1,tmp2 : CARDINAL;
    BEGIN
        IF (n = 1) OR (n = 0) THEN
            RETURN 1;
        ELSE
            tmp1 := Fibb(n-1);
            tmp2 := Fibb(n-2);
            RETURN (tmp1 + tmp2);
        END;
    END Fibb;
BEGIN
    Fibb(4);
END Fib.
```

Schets de stackstructuur op het moment in de tijd dat Fibb voor de eerste maal de waarde 1 als parameter meekrijgt ?

1.b. Functie

```
MODULE Fib
PROCEDURE Fibb(n:CARDINAL):CARDINAL;
    BEGIN
        IF (n = 1) OR (n = 0) THEN
            RETURN 1;
        ELSE
            RETURN (Fibb(n-1)+Fibb(n-2));
        END;
    END Fibb;
BEGIN
    Fibb(4);
END Fib.
```

Schets de stackstructuur op het moment dat Fibb voor de eerste maal de waarde 1 als parameter meekrijgt ?

1.c. Procedure met Call by variable/reference parameter

```
MODULE Fib
PROCEDURE Fibb(n:CARDINAL;VAR r : CARDINAL);
    VAR tmp1,tmp2 : CARDINAL;
    BEGIN
        IF (n = 1) OR (n = 0) THEN
            r := 1;
        ELSE
            Fibb(n-1,tmp1);
            Fibb(n-2,tmp2);
            r := tmp1 + tmp2;
        END;
    END Fibb;
BEGIN
    Fibb(4);
END Fib.
```

Schets de stackstructuur op het moment dat Fibb voor de eerste maal de waarde 1 als 1ste parameter meekrijgt ?