B38DF: Computer Architecture and Embedded Systems

Instructions
A Simple Processor
Assembly Coding Examples

Alexander Belyaev

Heriot-Watt University

School of Engineering & Physical Sciences

Electrical, Electronic and Computer Engineering

E-mail: a.belyaev@hw.ac.uk

Office: EM2.29

Based on materials prepared by Dr. Mustafa Suphi Erden and Dr. Senthil Muthukumaraswamy

Assembly Coding with a Six-Instruction Programmable Processor 1

Code the following in assembler (machine mnemonics)

D4 = D2 + D1 - D0

Load instruction—**MOV Ra, d**

- specifies the operation *RF[a]=D[d]*.

Store instruction—**MOV d, Ra**

- specifies the operation *D[d]=RF[a]*

Add instruction—**ADD Ra, Rb, Rc**

- specifies the operation *RF[a]=RF[b]+RF[c]*

Load-constant instruction—**0011** $\mathbf{r_3 r_2 r_1 r_0}$ $\mathbf{c_7 c_6 c_5 c_4 c_3 c_2 c_1 c_0}$

- **MOV Ra, #c**—specifies the operation *RF[a]=c*

Subtract instruction—**0100** $\mathbf{ra_3 ra_2 ra_1 ra_0}$ $\mathbf{rb_3 rb_2 rb_1 rb_0}$ $\mathbf{rc_3 rc_2 rc_1 rc_0}$

- **SUB Ra, Rb, Rc**—specifies the operation *RF[a]=RF[b] – RF[c]*

Jump-if-zero instruction—**0101** $\mathbf{ra_3 ra_2 ra_1 ra_0}$ $\mathbf{o_7 o_6 o_5 o_4 o_3 o_2 o_1 o_0}$

- **JMPZ Ra, offset**—specifies the operation *PC = PC + offset* if *RF[a]* is 0

Assembly Coding with a Six-Instruction Programmable Processor 1

Code the following in assembler (machine mnemonics)

D4 = D2 + D1 - D0

MOV R2, D2

MOV R1, D1

MOV R0, D0

MOV R4, #0

ADD R4, R1, R2

SUB R4, R4, R0

MOV D4, R4

Load instruction—**MOV Ra, d**

- specifies the operation *RF[a]=D[d]*.

Store instruction—**MOV d, Ra**

- specifies the operation *D[d]=RF[a]*

Add instruction—**ADD Ra, Rb, Rc**

- specifies the operation *RF[a]=RF[b]+RF[c]*

Load-constant instruction—**0011** $\mathbf{r_3r_2r_1r_0\ c_7c_6c_5c_4c_3c_2c_1c_0}$

- **MOV Ra, #c**—specifies the operation *RF[a]=c*

Subtract instruction—**0100** $\mathbf{ra_3ra_2ra_1ra_0\ rb_3rb_2rb_1rb_0\ rc_3rc_2rc_1rc_0}$

- **SUB Ra, Rb, Rc**—specifies the operation *RF[a]=RF[b] – RF[c]*

Jump-if-zero instruction—**0101** $\mathbf{ra_3ra_2ra_1ra_0\ o_7o_6o_5o_4o_3o_2o_1o_0}$

- **JMPZ Ra, offset**—specifies the operation *PC = PC + offset* if *RF[a]* is 0

Assembly Coding with a Six-Instruction Programmable Processor 2

Code the following in assembler (machine mnemonics) .

N is stored in D[9]

```
i = 0;
sum = 0;
while ( i != N ){
  sum = sum + i;
  i = i + 2;
}
```

Load instruction—**MOV Ra, d**

- specifies the operation *RF[a]=D[d]*.

Store instruction—**MOV d, Ra**

- specifies the operation *D[d]=RF[a]*

Add instruction—**ADD Ra, Rb, Rc**

- specifies the operation *RF[a]=RF[b]+RF[c]*

Load-constant instruction—**0011** $r_3r_2r_1r_0$ $c_7c_6c_5c_4c_3c_2c_1c_0$

- **MOV Ra, #c**—specifies the operation *RF[a]=c*

Subtract instruction—**0100** $ra_3ra_2ra_1ra_0$ $rb_3rb_2rb_1rb_0$ $rc_3rc_2rc_1rc_0$

- **SUB Ra, Rb, Rc**—specifies the operation *RF[a]=RF[b] – RF[c]*

Jump-if-zero instruction—**0101** $ra_3ra_2ra_1ra_0$ $o_7o_6o_5o_4o_3o_2o_1o_0$

- **JMPZ Ra, offset**—specifies the operation *PC = PC + offset* if *RF[a]* is 0

Assembly Coding with a Six-Instruction Programmable Processor 2

Code the following in assembler (machine mnemonics) .

N is stored in D[9]

```
i = 0;
sum = 0;
while ( i != N ){
   sum = sum + i;
   i = i + 2;
}
```

```
         MOV R0, #0     // R0 is "i"
         MOV R1, #0     // R1 is "sum"
         MOV R2, #2
         MOV R3, D[9]  // R3 is "N"
         MOV R4, D[9]  // for looping
         MOV R5, #0 // for looping
loop:    SUB R4, R3, R0 // R4 = N - i
         JMPZ R4, done
         ADD R1, R1, R0 // sum= sum+i
         ADD R0 R0, R2 //  i = i + 2
         JMPZ R5, loop
done:
```

Load instruction—**MOV Ra, d**

- specifies the operation *RF[a]=D[d]*.

Store instruction—**MOV d, Ra**

- specifies the operation *D[d]=RF[a]*

Add instruction—**ADD Ra, Rb, Rc**

- specifies the operation *RF[a]=RF[b]+RF[c]*

Load-constant instruction—**0011 $r_3r_2r_1r_0$ $c_7c_6c_5c_4c_3c_2c_1c_0$**

- **MOV Ra, #c**—specifies the operation *RF[a]=c*

Subtract instruction—**0100 $ra_3ra_2ra_1ra_0$ $rb_3rb_2rb_1rb_0$ $rc_3rc_2rc_1rc_0$**

- **SUB Ra, Rb, Rc**—specifies the operation *RF[a]=RF[b] – RF[c]*

Jump-if-zero instruction—**0101 $ra_3ra_2ra_1ra_0$ $o_7o_6o_5o_4o_3o_2o_1o_0$**

- **JMPZ Ra, offset**—specifies the operation *PC = PC + offset* if *RF[a]* is 0

Assembly Coding with a Six-Instruction Programmable Processor 3

Write a program to calculate
1+3+5+…+19

```
sum=0,term=1;
for(i=1; i<=10; i++)
{
  sum += term;
  term += 2;
}
```

Load instruction—**MOV Ra, d**

- specifies the operation *RF[a]=D[d]*.

Store instruction—**MOV d, Ra**

- specifies the operation *D[d]=RF[a]*

Add instruction—**ADD Ra, Rb, Rc**

- specifies the operation *RF[a]=RF[b]+RF[c]*

Load-constant instruction—**0011** $\mathbf{r_3r_2r_1r_0\ c_7c_6c_5c_4c_3c_2c_1c_0}$

- **MOV Ra, #c**—specifies the operation *RF[a]=c*

Subtract instruction—**0100** $\mathbf{ra_3ra_2ra_1ra_0\ rb_3rb_2rb_1rb_0\ rc_3rc_2rc_1rc_0}$

- **SUB Ra, Rb, Rc**—specifies the operation *RF[a]=RF[b] – RF[c]*

Jump-if-zero instruction—**0101** $\mathbf{ra_3ra_2ra_1ra_0\ o_7o_6o_5o_4o_3o_2o_1o_0}$

- **JMPZ Ra, offset**—specifies the operation *PC = PC + offset* if *RF[a]* is 0

Assembly Coding with a Six-Instruction Programmable Processor 3

Write a program to calculate 1+3+5+…+19

```
sum=0,term=1;
for(i=1; i<11; i++)
{ sum += term;
  term += 2;
}
```

Load instruction—**MOV Ra, d**

- specifies the operation *RF[a]=D[d]*.

Store instruction—**MOV d, Ra**

- specifies the operation *D[d]=RF[a]*

Add instruction—**ADD Ra, Rb, Rc**

- specifies the operation *RF[a]=RF[b]+RF[c]*

Load-constant instruction—**0011 $r_3r_2r_1r_0$ $c_7c_6c_5c_4c_3c_2c_1c_0$**

- **MOV Ra, #c**—specifies the operation *RF[a]=c*

Subtract instruction—**0100 $ra_3ra_2ra_1ra_0$ $rb_3rb_2rb_1rb_0$ $rc_3rc_2rc_1rc_0$**

- **SUB Ra, Rb, Rc**—specifies the operation *RF[a]=RF[b] – RF[c]*

Jump-if-zero instruction—**0101 $ra_3ra_2ra_1ra_0$ $o_7o_6o_5o_4o_3o_2o_1o_0$**

- **JMPZ Ra, offset**—specifies the operation *PC = PC + offset* if *RF[a]* is 0

```
.def sum = R0
.def term = R1
.def i = R2
.def tmp = R3
.def one = R4
.def N = R5
.def zero = R7
   MOV sum,#0
   MOV term,#1
   MOV i,#1
   MOV one,#1
   MOV N, #11
   MOV zero, #0
again:
   SUB  N,N,one
   JMPZ N,exit
   ADD sum,sum,term
   ADD term,term,one
   ADD term,term,one
   JMPZ zero,again
exit:
```

Assembly Coding with a Six-Instruction Programmable Processor 4

A programme is required to output the Fibonacci series. Using the formula y(n) = y(n-2) + y(n-1), where y(n) is the current number and y(n-1) and y(n-2) are previous two numbers of the series. Write a programme in machine mnemonics to output the first 10 numbers. Start from y(0) = 0 and y(1) = 1.

Leonardo Fibonacci

(born c. 1170, Pisa? - died after 1240)

```
1,1,2,3,5,8,13,21,35,55,…
Fnext = Fcurrent + Fprevious
```

Assembly Coding with a Six-Instruction Programmable Processor 4

A programme is required to output the Fibonacci series. Using the formula y(n) = y(n-2) + y(n-1), where y(n) is the current number and y(n-1) and y(n-2) are previous two numbers of the series. Write a programme in machine mnemonics to output the first 10 numbers. Start from y(0) = 0 and y(1) = 1.

1,1,2,3,5,8,13,21,35,55,…

Fnext = Fcurrent + Fprevious

```
Fp=0, Fc=1, tmp=0;
for(i=1; i<10; i++)
{
  tmp = Fc;
  Fc += Fp;
  Fp = tmp;
}
```

Leonardo Fibonacci

(born c. 1170, Pisa? - died after 1240)

Assembly Coding with a Six-Instruction Programmable Processor 4

A programme is required to output the Fibonacci series. Using the formula y(n) = y(n-2) + y(n-1), where y(n) is the current number and y(n-1) and y(n-2) are previous two numbers of the series. Write a programme in machine mnemonics to output the first 10 numbers. Start from y(0) = 0 and y(1) = 1.

```
Fp=0, Fc=1, tmp=0;
for(i=1; i<10; i++)
{
  tmp = Fc;
  Fc += Fp;
  Fp = tmp;
}
```

Load instruction—**MOV Ra, d**

- specifies the operation *RF[a]=D[d]*.

Store instruction—**MOV d, Ra**

- specifies the operation *D[d]=RF[a]*

Add instruction—**ADD Ra, Rb, Rc**

- specifies the operation *RF[a]=RF[b]+RF[c]*

Load-constant instruction—**0011** $\mathbf{r_3r_2r_1r_0}$ $\mathbf{c_7c_6c_5c_4c_3c_2c_1c_0}$

- **MOV Ra, #c**—specifies the operation *RF[a]=c*

Subtract instruction—**0100** $\mathbf{ra_3ra_2ra_1ra_0}$ $\mathbf{rb_3rb_2rb_1rb_0}$ $\mathbf{rc_3rc_2rc_1rc_0}$

- **SUB Ra, Rb, Rc**—specifies the operation *RF[a]=RF[b] – RF[c]*

Jump-if-zero instruction—**0101** $\mathbf{ra_3ra_2ra_1ra_0}$ $\mathbf{o_7o_6o_5o_4o_3o_2o_1o_0}$

- **JMPZ Ra, offset**—specifies the operation *PC = PC + offset* if *RF[a]* is 0

Assembly Coding with a Six-Instruction Programmable Processor 4

A programme is required to output the Fibonacci series. Using the formula y(n) = y(n-2) + y(n-1), where y(n) is the current number and y(n-1) and y(n-2) are previous two numbers of the series. Write a programme in machine mnemonics to output the first 10 numbers. Start from y(0) = 0 and y(1) = 1.

```
Fp=0, Fc=1, tmp=0;
for(i=1; i<10; i++)
{
  tmp = Fc;
  Fc += Fp;
  Fp = tmp;
}
```

```
    MOV tmp,#0
    MOV Fc,#1
    MOV Fp,#0
    MOV N, #11
    MOV one,#1
    MOV zero,#0
again:
    SUB  N,N,one
    JMPZ N,exit
    MOV tmp,Fc
    ADD Fc,Fc,Fp
    MOV Fp,tmp
    JMPZ zero,again
exit:
```

Assembly Coding with a Six-Instruction Programmable Processor 4

A programme is required to output the Fibonacci series. Using the formula y(n) = y(n-2) + y(n-1), where y(n) is the current number and y(n-1) and y(n-2) are previous two numbers of the series. Write a programme in machine mnemonics to output the first 10 numbers. Start from y(0) = 0 and y(1) = 1.

```
Fp=0, Fc=1, tmp=0;
for(i=1; i<10; i++)
{
   tmp = Fc;
   Fc += Fp;
   Fp = tmp;
}
```

A different set of instructions and different assembly language are used:

Register

R00=	0x00
R01=	0x00
R02=	0x00
R03=	0x00
R04=	0x00
R05=	0x00
R06=	0x00
R07=	0x00
R08=	0x00
R09=	0x00
R10=	0x00
R11=	0x00
R12=	0x00
R13=	0x00
R14=	0x00
R15=	0x00
R16=	0x22
R17=	0x37

```
.def  tmp = r16
.def  Fc = r17
.def  Fp = r18
.def  i = r19
        ldi  tmp,0
        ldi  Fc,1
        ldi  Fp,0
        ldi  i,1
again:
        cpi  i,10
        brge exit
        mov  tmp,Fc
        add  Fc,Fp
        mov  Fp,tmp
        inc  i
        rjmp again
exit:
        nop
```