Arquitetura de Computadores

ISA – Instruções de controlo de fluxo

Estruturas programáticas de decisão e de ciclo
Bib: A – Secções 6.3.2 a 6.3.5

João Pedro Patriarca (jpatri@cc.isel.ipl.pt), Gabinete F.0.23 do edifício F

ISEL, ADEETC, LEIC

Instruções de controlo de fluxo

- Conjunto de instruções e respetivas codificações: manual de consulta rápida das instruções do P16
- Instruções que promovem saltos no programa. Os saltos podem ser:
  - Condicionais: a realização do salto depende de uma ou mais *flags* produzidas pela ALU na execução da última instrução de processamento de dados
  - Incondicionais: o salto realiza-se sempre não dependendo de qualquer *flag*

- O salto corresponde a um *offset* que é somado ao valor atual do registo PC
  - O *offset* representa um número relativo (inteiro com sinal)
  - O *offset* codificado na instrução corresponde ao número de instruções a saltar

```
    L0:
2000   bne  L1
2002   add  r0, r0, 1
    L1:
2004   sub  r1, r1, 1
2006   cmp  r0, r1
2008   b    L0
```

15 … 0

0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	1

(+1)

15 … 0

0	1	0	1	1	0	1	1	1	1	1	1	1	0	1	1

(-5)

- Instruções fundamentais para a implementação de estruturas de decisão e de ciclo

Mnemónicas P16 para instruções de salto condicionais

- BEQ/BZS – **B**ranch if **Eq**ual / **B**ranch if **Z**ero **S**et
- BNE/BZC – **B**ranch if **N**ot **E**qual / **B**ranch if **Z**ero **C**lear
- BCS/BLO – **B**ranch if **C**arry **S**et / **B**ranch if **Lo**wer
- BCC/BHS – **B**ranch if **C**arry **C**lear / **B**ranch if **H**igher or **S**ame
- BGE – **B**ranch if **G**reater or **E**qual
- BLT – **B**ranch if **L**ess **T**han

Comparação de números

	Operação (r0 = a, r1 = b)	Números naturais	Números relativos
`if ( a < b )`	`cmp r0, r1`	`bhs/bcc`	`bge`
`if ( a >= b )`	`cmp r0, r1`	`blo/bcs`	`blt`
`if ( a > b )`	`cmp r1, r0`	`bhs/bcc`	`bge`
`if ( a <= b )`	`cmp r1, r0`	`blo/bcs`	`blt`

`uint8_t`	Número natural a 8 bits
`uint16_t`	Número natural a 16 bits
`int8_t`	Número relativo a 8 bits
`int16_t`	Número relativo a 16 bits

Estrutura programática IF e IF/ELSE

```
uint16_t a;
…
if (a != 0)
   a >>= 2;
```

```
; a => r0
        …
        add     r0, r0, #0
        beq     if_end
        lsr     r0, r0, #2
if_end:
```

```
uint16_t a, b, c;
…
if (a < b)
   c = a * 2;
else
   c <<= 2;
```

```
; a => r0 , b => r1, c => r2
        …
        cmp     r0, r1
        bcc     if_else
        lsl     r2, r0, #1
        b       if_end
if_else:
        lsl     r2, r2, #2
if_end:
```

Estrutura programática SWITCH/CASE

```
uint16_t a;
int16_t b;
…
switch (a) {
   case 0: b *= 4; break;
   case 1: b /= 2; break;
   default: b = 0;
}
```

```
; a => r0, b => r1
        mov     r2, #2
        cmp     r0, r2
        bcc     switchcase_default
        lsl     r2, r0, #1
        add     pc, r2, pc
switchcase_table:
        b       switchcase_case0
        b       switchcase_case1
switchcase_case0:
        lsl     r1, r1, #2
        b       switchcase_end
switchcase_case1:
        asr     r1, r1, #1
        b       switchcase_end
switchcase_default:
        eor     r1, r1, r1
test_switchcase_end:
```

Estrutura programática DO-WHILE e WHILE

```
uint16_t a;
int16_t b;
…
do {
   a *= 2;
   b -= 1;
} while (b > 10);
```

```
; a => r0, b => r1
        mov     r2, #10
dowhile:
        lsl     r0, r0, #1
        sub     r1, r1, #1
        cmp     r2, r1
        blt     dowhile
```

```
int16_t a;
int8_t b;
uint16_t c;
…
while ( a <= b ) {
   c += a;
   b -= 1;
}
```

```
; a => r0, b => r1, c => r2
        lsl     r1, r1, #8
        asr     r1, r1, #8
        b       whilecond
while:
        add     r2, r2, r0
        sub     r1, r1, #1
whilecond:
        cmp     r1, r0
        bge     while
```

Estrutura programática FOR

```
uint16_t n;
…
for (uint16_t i = 0, a = 1; i <= n; ++i)
{
   a <<= 1;
}
```

```
; n => r0, a => r1, i => r2
        …
for_init:
        mov     r2, #0
        mov     r1, #1
        b       for_cond
for:
        lsl     r1, r1, #1
        add     r2, r2, #1
for_cond:
        cmp     r0, r2
        bhs     for
```

Desafios

1. Escreva na variável *abs* o valor absoluto da variável *v*. A variável *abs* está mapeada no registo R0 e *v* mapeada no registo R1. Ambas as variáveis representam valores inteiros com sinal a 16 bits
2. Escreva na variável *c* o maior valor entre as variáveis *a* e *b*. A variável *a* está mapeada no registo R0, *b* no registo R1 e *c* no registo R2. As três variáveis representam inteiros com sinal de 8 bits. Note que o bit de sinal das três variáveis é representado pelo oitavo bit, bit de maior peso. No entanto, a ALU realiza operações a 16 bits.
3. Escreva nas variáveis *q* e *r* o quociente e o resto resultantes da divisão de *x* por *y*. Todas as variáveis representam inteiros naturais expressos a 16 bits. Assuma os registos R0, R1, R2 e R3 para mapear as variáveis *x*, *y*, *q* e *r*, respetivamente.

Desafio 1 de 3

1. Escreva na variável *abs* o valor absoluto da variável *v*. A variável *abs* está mapeada no registo R0 e *v* mapeada no registo R1. Ambas as variáveis representam valores inteiros com sinal a 16 bits

```
int16_t abs, v;
…
if (v < 0)
   abs = -v;
else
   abs = v;
```

```
; abs => r0, v => r1
        …
        sub     r1, r1, #0
        bge     else
        add     r0, r1, #0
        b       end_if
if_else:
        eor     r0, r0, r0
        sub     r0, r0, r1
if_end:
```

Desafio 2 de 3

2. Escreva na variável *c* o maior valor entre as variáveis *a* e *b*. A variável *a* está mapeada no registo R0, *b* no registo R1 e *c* no registo R2. As três variáveis representam inteiros com sinal de 8 bits. Note que o bit de sinal das três variáveis é representado pelo oitavo bit, bit de maior peso. No entanto, a ALU realiza operações a 16 bits.

```
int8_t a, b, c;
…
if (a < b)
   c = b;
else
   c = a;
```

```
; a => r0, b => r1, c => r2
        lsl     r0, r0, #8 ; 8->16 nos Z
        asr     r0, r0, #8 ; 8->16 nos Z
        lsl     r1, r1, #8 ; 8->16 nos Z
        asr     r1, r1, #8 ; 8->16 nos Z
        cmp     r0, r1
        bge     if_else
        add     r2, r1, #0
        b       end_if
if_else:
        add     r2, r0, #0
if_end:
```

Desafio 3 de 3

3. Escreva nas variáveis *q* e *r* o quociente e o resto resultantes da divisão de *x* por *y*. Todas as variáveis representam inteiros naturais expressos a 16 bits. Assuma os registos R0, R1, R2 e R3 para mapear as variáveis *x*, *y*, *q* e *r*, respetivamente.

```
uint16_t x, y, q, r;
…
q = 0;
r = x;
if (y != 0)
   while (r >= y) {
      q += 1;
      r -= y;
   }
```

```
; x => r0, y => r1, q => r2, r => r3
        eor     r2, r2, r2
        add     r3, r0, #0
        and     r1, r1, r1
        bzs     if_end
while:
        cmp     r3, r1
        blo     if_end
        add     r2, r2, #1
        sub     r3, r3, r1
        b       while
if_end:
```