8
Introdução à Micro-Programação

Calvin: Queres ver uma coisa estranha? Repara, colocas pão nesta ranhura e carregas no botão para baixo. Esperas alguns minutos, e zás! Sai uma tosta!
Hobbes: Uau! Para onde foi o pão?
Calvin: Não faço a mínima ideia. Não é esquisito?

Calvin, em "Calvin & Hobbes" de Bill Watterson

Tal como o Calvin, temos estado a programar (até agora) em Assembly, sem saber o que se passa no interior da nossa "torradeira" que é o MIPS. Sabemos que o Assembly é uma linguagem de programação de baixo nível e que constitui uma representação simbólica da codificação binária de um computador: a linguagem máquina (ver Capítulo 1).

Ora, a linguagem máquina é composta por micro-instruções que indicam que operação digital deve o computador fazer. Cada instrução máquina é composta por um conjunto ordenado de zeros e uns, estruturado em campos. Cada campo contém a informação que se complementa para indicar ao processador que acção realizar.

Vamos então neste capítulo observar alguns exemplos dos programas em linguagem máquina correspondentes a alguns dos troços de programas Assembly que temos vindo a estudar. E ficaremos a perceber o que se passa "dentro" do processador.

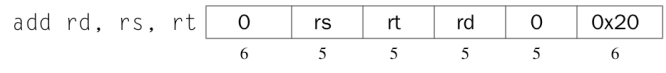
Formatos das instruções MIPS em linguagem máquina

Qualquer instrução MIPS codifica-se em linguagem máquina, isto é, zeros e uns. Uma instrução MIPS ocupa, como sabemos, 4 palavras. Isto significa que qualquer instrução MIPS será codificada usando 32 bits (mesmo que não necessite de todos esses bits).

Existem, como vimos, duas grandes classes de instruções Assembly MIPS: instruções implementadas em hardware (dizem-se *hard-wired*) e as pseudo-instruções implementadas pelo Assembler. Uma pseudo-instrução expande-se em mais de uma instrução *hard-wired*.

Uma instrução é codificada em vários campos (grupos de bits contíguos), cada um deles com um significado próprio. Por exemplo, a seguinte figura (retirada do manual do SPIM), ilustra a codificação da instrução add (com *overflow*):

Addition (with overflow)



Na adição com overflow, a instrução é codificada em seis campos. Cada campo tem uma largura (em termos de número de bits) indicada sob o respectivo campo. Esta instrução começa com um grupo a zeros (6 bits a zero, à esquerda). Este grupo de bits é o *opcode* (código da instrução).

Os registos são especificados por rs, rt e rd. Portanto o campo seguinte é especifica um registo chamado rs (de *register source*). Este registo é o segundo operando da instrução. Outro campo comum é designado imm16, que especifica um número imediato de 16

bits. De um modo geral, as instruções seguem o formato especificado na tabela seguinte:

Nome do campo	Cód. Instrução	2º Operando	3º Operando	1º Operando	shamt	Funct
Largura do Campo	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

R	Cód. da Inst. **6 bits**	rs **5 bits**	rt **5 bits**	rd **5 bits**	Shamt **5 bits**	Funct **6 bits**
I	Cód. da Inst. **6 bits**	rs **5 bits**	rt **5 bits**	Endereço Imediato **16 bits**		
J	Cód. da Inst. **6 bits**	Endereço **26 bits**				

A segunda tabela mostra que existem 3 tipos de formatos para as instruções: R, I e J, consoante o primeiro operando seja um registo (formato R), um imediato (formato I) ou a instrução seja uma instrução de salto (formato J), a qual só contém como argumento um endereço.

Q8.1. Partindo do seguinte programa simbólico escrito em Assembly do MIPS R2000, pretendemos obter o programa respectivo em linguagem máquina:

```
add $1, $2, $3          # Equivale a $1 # =$2+$3
sub $1, $2, $3          # Equivale a $1 # =$2-$3
lw $1, 100($2)          # Equivale a $1= Memória ($2+100)
sw $1, 100($2)          # Memória ($2+100) = $1
```

A primeira tabela apresenta o formato da instrução em causa, o seu código, os operandos e shamt/Funct. A segunda tabela é equivalente à anterior, mas mostra a codificação binária da instrução, ou seja, a instrução tal e qual como o computador a vê. E mostra também a instrução em hexadecimal (a qual também surge no SPIM).

	Cód. Instrução	1ºoperando	2ºoperando	3ºoperando	shamt	funct	Instrução MIPS
Formato R	0	2	3	1	0	32	add $1,$2,$3
Formato R	0	2	3	1	0	34	sub $1,$2,$3
Formato I	35	2	1	100			lw $1,100($2)
Formato I	43	2	1	100			sw $1,100($2)

	Cód. Instrução	2º Operando	3º Operando	1º Operando	shamt	Funct	
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	
				ou 16 bits			em Hexadecimal:
em Binário:	0000 00	00 010	0 0011	0000 1	000 00	10 0000	0x00430820
	0000 00	00 010	0 0011	0000 1	000 00	10 0010	0x00430822
	1000 11	00 010	0 0001	0000 0000 0011 0100			0x8c410034
	1010 11	00 010	0 0001	0000 0000 0011 0100			0xac410034

Q8.2. Considere o seguinte código escrito em linguagem Assembly do processador MIPS:

```
la $t0,palavra # palavra corresponde ao endereço 0x10010005 do seg. dados
lw $a0,0($t0)
jal randnum    # randnum reside no endereço 0x00400060 do seg. código
add $t2,$v0,$0
```

Sabendo que a primeira instrução tem como endereço `0x400028`, traduza este troço de código para linguagem máquina MIPS e escreva o conteúdo de cada instrução em binário e hexadecimal numa tabela com o seguinte aspecto:

Endereços	Cód. Instrução	2º Operando	3º Operando	1º Operando	shamt	Funct	
0x00400028							lui $8, 0x1001

Endereços	Cód. Instrução	2º Operando	3º Operando	1º Operando	shamt	Funct	
Largura do Campo	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	em Hexadecimal:
0x00400028							0x ...

Resolução:

(há que ter em atenção que a instrução `la` é uma instrução que se expande em duas – neste caso, uma vez que o endereço de `palavra` tem de ser calculado).

Endereços	Cód. Instrução	rs	rt	rd	shamt	Funct	
0x00400028	15	0	8	0x1001			lui $8,0x1001
0x0040002c	8	8	8	5			addi $8,$0,5
0x00400030	35	8	4	0			lw $4,0($8)
0x00400034	3	0x400060					jal 0x400060
0x00400038	0	2	0	10	0	32	add $10,$2,$0

Endereços	Cód. Instrução	2º Operando	3º Operando	1º Operando	shamt	Funct	
Largura do Campo	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	em Hexadecimal:
0x00400028	0011 11	00 000	0 1000	0001 0000 0000 0001			0x3c081001
0x0040002c	0010 00	01 000	0 1000	0000 0000 0000 0101			0x21080005
0x00400030	1000 11	01 000	0 0100	0000 0000 0000 0000			0x8d040000
0x00400034	0000 11	00 0001 0000 0000 0000 0001 1000					0x0c100018
0x00400038	0000 00	00 010	0 0000	0101 0	000 00	10 0000	0x00405020

Q8.3. Pretende-se o programa em Assembly do MIPS para a seguinte instrução em pseudo-código: `A[12] = h + A[8]`

Assumindo que A é uma tabela e associando a variável h ao registo `$s1`, e sabendo ainda que o endereço base de `A` encontra-se em `$s3`:

```
main:   lw $t0,32($s3)        # carrega A[8] em $t0
        add $t0,$s1,$t0       # soma $t0=h+A[8]
        sw $t0,48($s3)        # armazena $t0 em A[12]
```

A tabela seguinte mostra este programa em linguagem máquina:

Endereços	Cód. Instrução	2º Operando	3º Operando	1º Operando	shamt	Funct	
0x00400000	35	19	8	8*4 = 32			lw $t0,32($s3)
0x00400004	0	17	8	8	0	32	add $t0,$s1,$t0
0x00400008	43	19	8	12*4 = 48			sw $t0,48($s3)

E esta tabela mostra o programa em hexadecimal:

Endereços	Cód. Instrução	2º Operando	3º Operando	1º Operando	shamt	Funct	
Largura do Campo	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	em Hexadecimal:
0x00400000	1000 11	10 011	0 1000	0000 0000 0010 0000			0x8e680020
0x00400004	0000 00	10 001	0 1000	0100 0	000 00	10 0000	0x02284020
0x00400008	1010 11	10 011	0 1000	0000 0000 0011 0000			0xae680030

Q8.4. Considere o seguinte segmento de programa em linguagem C:

```
if (i==j)
   h=i+j;
else
   h=i-j;
```

Escreva o respectivo troço de programa em linguagem Assembly do MIPS. Em seguida, preencha as tabelas com os formatos apresentados (programa em linguagem máquina e em hexadecimal). Considere que tem as variáveis h, i e j nos registos $16, $17 e $18, em que se $17=$18 somam-se; se diferentes, subtraem-se os conteúdos dos registos $17 e $18.

Resolução:

O programa Assembly pedido é muito simples:

```
main:   bne $17, $18, L1
        add $16, $17, $18
        j Exit
L1:     sub $16, $17, $18
Exit:   jr $ra
```

O programa em linguagem máquina é o seguinte:

Endereços	Cód. Instrução	2º op.	3º op.	1º op.	shamt	Funct	
0x00400000	5	17	18	3*4=12			bne $17,$18,L1
0x00400004	0	17	18	16	0	32	add $16,$17,$18
0x00400008	3	0x400010					j Exit
0x0040000c	0	17	18	16	0	34	sub $16,$17,$18
0x00400010	0	31	0	0	0	8	Exit: jr $ra

Questão: qual o formato da instrução `bne`? Porque é que o último campo tem o valor 12 (= 3*4)?