Arquitetura de Computadores 2022/23

TPC 1

Entrega: 23h59 de 17 de abril de 2023

Este trabalho de casa consiste num exercício de programação **a ser realizado em grupo de no máximo dois alunos**. Pode esclarecer dúvidas gerais com colegas, mas a solução e a escrita do código deve ser estritamente realizada pelos membros do grupo. Todas as resoluções serão comparadas de forma automática e os casos de plágio serão punidos de acordo com os regulamentos em vigor.

A forma de entrega será divulgada oportunamente, usando uma mensagem via CLIP.

Exercício

Neste exercício deve completar o código de um simulador (escrito na linguagem C) de uma arquitetura composta por um CPU muito simples e uma memória. Este inclui, já implementada, uma consola onde é possível um utilizador dar comandos para ler e escrever na memória central, incluindo ler para memória os valores representados num ficheiro de texto. Pode ainda mandar executar as instruções presentes na memória. Os comandos implementados são os seguintes:

> `poke` *EEE X* – escreve no endereço *EEE* o valor *X*.
>
> `peek` *EEE* – ler o valor no endereço de memória *EEE*. O valor é afixado em hexadecimal.
>
> `dump` *X* – mostra todo o conteúdo dos registos e de *X* palavras da memória a partir do endereço 0.
>
> `load` *FILE* – lê o ficheiro de nome *FILE*, interpretando cada linha de texto como um valor que é colocado sequencialmente na memória.
>
> `run` – executa as instruções a partir do endereço 0 de memória.

Os valores *X* e *EEE* podem ser dados em base 10 ou em base 16 se iniciados por `0x`.

O código a completar está no ficheiro `dorun.c` e deve implementar o ciclo de *fetch*, *decode* e *execute* para simular a execução das instruções na memória. A execução deve decorrer sempre a partir do endereço 0 de memória, até à execução da instrução HALT (ver secção seguinte).

Instruções máquina do processador:

O processador usa palavras de 32 bits e tem 16 registos (AC). Tem ainda um registo como *Program Counter* (PC) e outro para a instrução a executar (IR). A memória está também organizada em palavras de 32 bits por endereço, sendo cada endereço de 12 bits. Há também duas *flags*

- ZERO contém um valor diferente de zero quando a última operação aritmética deu zero e 0 quando a última operação aritmética não deu zero
- POSITIVO contêm um valor diferente de zero quando a última operação aritmética deu um resultado >= 0 e zero quando a última operação aritmética deu um resultado < 0

As instruções têm tamanho fixo de 32 bits, sendo os 8 mais significativos para o código de operação, 12 bits para especificar os registos que contêm os dois operandos e o resultado de instruções aritméticas e os restantes 12 para o endereço, se existir.

31 24	23 20	19 16	15 12	11 0
Código da instrução	Registo 1	Registo 2	Registo 3	Endereço

Bits 31-24: Código da instrução: especifica a instrução a executar e como devem ser interpretados os bits 23 a 0

Bits 23-20: Registo 1(fonte 1 (src1): especifica qual o registo cujo conteúdo vai ser o operando 1 da instrução aritmética e lógica especificada nos bits 31 a 24.

Bits 19-16: Registo 2 (fonte 2 (src2): especifica qual o registo cujo conteúdo vai ser o operando 2 da instrução aritmética e lógica especificada.

Bits 15-12: Registo 3 (destino (dst): especifica qual o registo cujo conteúdo vai ter o resultado da instrução aritmética e lógica especificada**.**

Bits 11-0: endereço: especifica o endereço de memória a usar nas instruções que a referenciam (load, store, saltos)

Há exceções a esta organização e em algumas instruções há bits que não são considerados. As instruções suportadas e o respetivo código máquina em representação hexadecimal, são descritas a seguir. Na coluna *Instrução* os valores apresentados são sequências de dígitos em base 16 (hexadecimal) cada um correspondendo a 4 bits. São usadas as convenções:

Símbolo	Significado
X	4 bits a ignorar
EEE	Endereço com 12 bits; é um inteiro sem sinal
reg	4 bits que especificam um registo (0x0 a 0xF)
VVVVV~~V~~ 20	~~24~~ bits que especificam um valor codificado em complemento para 2 ou seja um inteiro com sinal que pode representar valores de $2^{~~23~~}$ -1 a $-2^{~~23~~}$

Instrução	Mnemónica	Descrição
00XXXXXX	HALT	Termina o programa
01regVVVVV	LDI reg valor	O registo especificado nos bits 23 a 20 recebe o valor VVVVV; os bits 31 a 20 do registo recebem 0 se o bit 23 for 0 e 1 se o bit 23 for 1 (extensão do sinal)
02regXXEEE	LOAD reg endereço	O registo especificado nos bits 23 a 20 recebe o conteúdo do endereço EEE que está especificado nos bits 11-0. Os bits 19 a 12 são ignorados.
03regXXEEE	STORE reg endereço	O conteúdo do registo especificado nos bits 23 a 20 é guardado na posição de memória cujo endereço EEE está especificado nos bits 11-0. Os bits 19 a 12 são ignorados.
04reg1reg2reg3XXX	ADD reg1 reg2 reg3	O registo 3 (especificado nos bits 15 a 12) recebe a soma do conteúdo do registo especificado nos bits 23 a 20 com o conteúdo do registo especificado nos bits 19 a 16. Os bits 11 a 0 são ignorados.
05reg1reg2reg3XXX	SUB reg1 reg2 reg3	O registo 3 (especificado nos bits 15 a 12) recebe a subtração do conteúdo do registo especificado nos bits 23 a 20 com o conteúdo do registo especificado nos bits 19 a 16. Os bits 11 a 0 são ignorados.
06regXXXXXX	CLEAR reg	O conteúdo do registo especificado nos bits 23 a 20 passa a ser 0.
08XXXXEEE	JUMP EEE	O PC recebe o valor EEE que está nos bits 11 a 0. Os bits 24 a 12 são ignorados.
09XXXXEEE	JZ EEE	O PC recebe o valor EEE que está nos bits 11 a 0, se o código de condição (flag) ZERO for True (à moda do C). Os bits 24 a 12 são ignorados, assim como nas instruções seguintes.
0AXXXXEEE	JNZ EEE	O PC recebe o valor EEE que está nos bits 11 a 0, se o código de condição (flag) ZERO for False (à moda do C).
0BXXXXEEE	JG EEE	O PC recebe o valor EEE que está nos bits 11 a 0, se o resultado da última subtração deu um número maior do que 0. Isto quer dizer que ZERO é False e POSITIVO é True.
0CXXXXEEE	JGE EEE	O PC recebe o valor EEE que está nos bits 11 a 0, se o resultado da última subtração deu um número maior ou igual a 0. Isto quer dizer que POSITIVO é True.
0DXXXXEEE	JB EEE	O PC recebe o valor EEE que está nos bits 11 a 0, se o o resultado da última subtração deu um número menor do que 0. Isto quer dizer que ZERO é False e POSITIVO é False.
0EXXXXVVV	JBE EEE	O PC recebe o valor EEE que está nos bits 11 a 0, se o o resultado da última subtração deu um número menor ou igual a 0. Isto quer dizer que ou ZERO é true ou POSITIVO é False.

Exemplo de um programa para multiplicar dois inteiros sem sinal (como exemplo, calcular 2x3). Os bits irrelevantes estão colocados a 0. Note-se que no ficheiro a carregar no simulador apenas pode estar a coluna do meio, sem o cabeçalho.

```
end.    code          mnemónica
0x00:   0x02100010    LOAD r1 0x10 // r1 ← multiplicando
0x01:   0x02200011    LOAD r2 0x11 // r2 ← multiplicador
0x02:   0x01300001    LOADI r3 1  // r3 ← 1(para decrementar)
0x03:   0x06400000    CLEAR r4    // r4 ← 0 (para as comparações)
0x04:   0x06500000    CLEAR r5    // r5 ← 0 (acumular o produto)
0x05:   0x05242000    SUB r2 r4 r2 (r2 é 0 ?)
0x06:   0x0900000A    JZ 0x00A
0x07:   0x04155000    ADD r1 r5 r5
0x08:   0x05232000    SUB r2 r3 r2     //  r2 ← r2 - 1
0x09:   0x08000005    JMP 0x05
0x0A:   0x03500012    STORE r5 0x12
0x0B:   0x00000000    HALT

0x10:   2    (multiplicando  preeenchido com poke)
0x11:   3    (multiplicador  preenchido poke)
0x12;   0    (resultado a ler no final com peek)
```

O ficheiro `p.code` fornecido contém o código e dados deste programa em notação hexadecimal, ocupando uma palavra de memória por linha. Correndo o simulador, podemos carregar o programa fazendo `load p.code` e executar com `run`. Podemos consultar o resultado vendo o que fica na variável de endereço 0x12 com `peek`. Podemos alterar as variáveis correspondentes ao multiplicando e multiplicador usando o comando `poke`. Exemplo duma sessão para calcular 4x2:

```
cmd> load p.code

cmd> poke 0x10 4
cmd> poke 0x11 2
cmd> run
HALT instruction executed
cmd> peek 0x12
0x12: 0x8
```

Entrega

A entrega faz-se através de um e-mail dirigido ao docente do seu turno prático:

- P1, P3, P5, P7 Professora Maria Cecília Gomes (mcg@fct.unl.pt)
- P2, P4, P6, P8 Professor Kevin Gallagher (k.gallagher@fct.unl.pt)
- P9, P10 Monitor Henrique Campos Ferreira (hjp.ferreira@campus.fct.unl.pt)

A entrega deve ser feita dentro do prazo, submetendo apenas o seu ficheiro *dorun.c.* O cumprimento da especificação pelo código entregue não é o único critério para definir a nota. Outros critérios, como a qualidade do código e completude da solução, também são tidos em conta.

O programa será compilado com o seguinte comando:

```
cc -Wall -std=c11 -o sim sim.c dorun.c
```