Manual do Assemblador e Simulador do Processador P4

Dinis Pedro Pinto Marcos Madeira

com contribuições adicionais do corpo docente de Introdução à Arquitetura de Computadores – LEIC-A

Instituto Superior Técnico - Universidade de Lisboa

dinismadeira@tecnico.ulisboa.pt

Conteúdo

1 Introdução

Este documento descreve o funcionamento do assemblador e simulador do Pequeno Processador Pedagógico com *Pipeline* (P4).

O assemblador e simulador encontram-se integrados numa única solução disponível sob a forma de uma aplicação para Windows, Linux e Mac, e também como uma versão online que corre diretamente num navegador web.

O assemblador é constituído por um editor de texto que permite introduzir o código Assembly e gera automaticamente o código binário correspondente. As linhas de código que não possam ser convertidas em código binário serão assinaladas com a indicação do erro. É possível gerar um ficheiro p4z que consiste num arquivo zip com os ficheiros de inicialização de memória (MIF) necessários para correr o programa numa placa com o P4.

O simulador permite executar o programa emulando as entradas e saídas disponíveis na placa do P4. É possível definir a velocidade máxima do relógio, bem como executar passo a passo, ou definir pontos de paragem que irão parar a simulação imediatamente antes da execução das instruções assinaladas.

2 Arquitetura do Processador P4

2.1 Registos

O processador P4 contém 8 registos genéricos, R0 a R7. A descrição destes registos é apresentada na Tabela 1.

O registo R0 tem sempre o valor 0 e qualquer escrita para este registo não terá efeito.

Os registos são inicializados a 0 após a reinicialização do processador.

Registo	Descrição	Preservado por chamada a função?
`R0`	O valor do registo R0 é sempre 0. Qualquer operação de escrita não terá qualquer efeito no valor do registo.	N/A
`R1-R2`	Registos para passagem de argumentos das funções. Caso haja mais argumentos, estes devem ser passados pela pilha.	Não
`R3`	Registo para o valor de retorno de funções	Não
`R4-R5`	Registos de uso geral	Sim
`R6`	Apontador para a pilha. Contém o endereço da última posição preenchida da pilha.	Sim
`R7`	Endereço de retorno	Não

Tabela 1. Descrição dos registos do P4 e convenção de chamada de funções.

2.2 Convenção de chamada

O processador P4 é um processador *Reduced Instruction Set Computer* (RISC) e, como tal, segue a convenção de chamada habitual deste tipo de processadores. Ao chamar funções, os parâmetros das

funções são passados utilizando os registos dedicados mencionados na Tabela 1, `R1` e `R2`. Caso haja mais de dois parâmetros de entrada, os parâmetros adicionais devem ser passados pela pilha. O retorno da função é feito utilizando o registo para o efeito, `R3`.

Entre a chamada a uma função o retorno da mesma, o código da função deve preservar o valor dos registos `R4-R6`[1]. Caso seja necessário modificar os registos R4 e R5 para programar a função, os seus valores deverão ser salvaguardados na pilha, à entrada da função, e os seus valores originais deverão ser repostos à saida da função (retirando os valores respetivos da pilha).

2.3 Pilha

A pilha cresce no sentido de endereços decrescentes e o apontador para a pilha, `R6`, aponta para a última posição preenchida.

A Figura 1 apresenta um exemplo do crescimento de pilha, com base no endereço `0x7ff8`. Neste exemplo, a pilha tem duas posições preenchidas apenas, `0x7ff8` e `0x7ff7`, com o apontador da pilha, `R6`, a apontar para a última posição preenchida, `0x7ff7`.

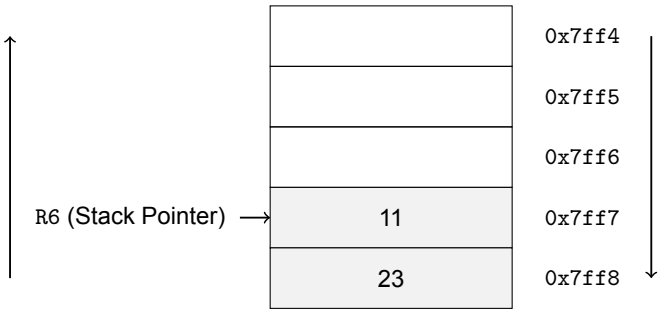

Figura 1. Exemplo de crescimento da pilha.

2.4 Bits de Estado

Existem 5 bits de estado, designados pelas letras E, Z, C, N e O, guardados num registo, RE. O significado de cada um destes bits é:

- O: *overflow* ou excesso. É definido pelas operações aritméticas e indica que o resultado está incorreto quando interpretado em complemento para 2 por exceder o número de bits disponíveis no registo de destino.
- N: *negative* ou sinal. É definido pelas operações aritméticas, lógicas e de deslocamento. Indica que o resultado da operação corresponde a um número negativo quando interpretado em complemento para 2, ou seja, o bit mais significativo tem valor 1.

[1] Exceptua-se contudo o caso de funções que devolvam mais do que um valor. Nesse caso, o primeiro valor deverá ser devolvido por registo, e os restantes pela pilha, obrigando à modificação do registo R6.

- C: *carry* ou transporte. É definido pelas operações aritméticas e de deslocamento. Nas operações aritméticas indica que o resultado pode estar incorreto quando interpretado como um número sem sinal por exceder o número de bits disponíveis no registo de destino. Nas operações de deslocamento corresponde ao bit deslocado para fora por estas operações. Pode ser modificado através das instruções `STC`, `CLC` e `CMC`.
- Z: zero. É definido pelas operações aritméticas, lógicas e de deslocamento, e indica que o resultado da operação foi zero.
- E: *enable interrupts*. Qualquer rotina de serviço de uma interrupção só será chamada quando este bit tiver o valor 1. Este bit pode ser modificado através das instruções `ENI` e `DSI`.

2.5 Memória

O P4 possui duas memórias independentes, a memória de programa e a memória de dados. Cada memória possui 32k palavras de 16 bits.

A memória de programa contém as instruções do programa e não pode ser acedida pelas instruções assembly.

A memória de dados pode ser lida e escrita através das instruções `LOAD` e `STOR`, respetivamente.

Nos acessos a estas memórias, apenas os 15 bits menos significativos do endereço são tidos em conta. Isto significa, por exemplo, que um valor escrito para o endereço 8000h será colocado na primeira posição da memória de dados, ou que o endereço FE00h é equivalente ao endereço 7E00h.

2.6 Entradas e Saídas

O espaço de entradas e saídas é mapeado em memória. Os endereços de memória a partir de FF00h estão reservados para o espaço de entradas e saídas. O acesso aos dispositivos de entrada/saída mapeados neste espaço pode ser feito através das instruções `LOAD` e `STOR`.

Os dispositivos de entrada/saída disponíveis no P4 são:

Terminal.

Consiste numa interface de janela de texto e teclado. Possui os seguintes portos:

Endereço FFFFh: leitura, permite receber caracteres introduzidos através do teclado.

Endereço FFFEh: escrita, permite escrever caracteres na janela de texto.

Endereço FFFDh: estado, permite testar se houve uma tecla premida.

Endereço FFFCh: controlo, permite posicionar o cursor na janela de texto.

Endereço FFFBh: cor, permite definir a cor do texto e do fundo.

Interruptores.

Endereço FFF9h: permite obter o estado dos 10 interruptores do P4.

LED.

A placa do P4 possui 10 LED que podem ser controlados individualmente.

Endereço FFF8h: permite ligar/desligar os LED.

Ecrã LCD.

Ecrã LCD com 2 linhas de 16 colunas.

Endereço FFF5h: escrita, permite escrever caracteres no ecrã.

Endereço FFF4h: controlo, permite posicionar o cursor no ecrã.

Temporizador

É possível definir um temporizador que gera uma interrupção após um intervalo de tempo real definido pelo utilizador.

Endereço FFF7h: controlo, permite controlar o funcionamento do temporizador. Quando o bit menos significativo do valor escrito é 1, o temporizador inicia a contagem decrescente, caso seja 0, o temporizador pára a contagem.

Endereço FFF6h: valor de contagem, define o valor da contagem, que corresponde à duração do temporizador em décimas de segundo.

Mostradores Hexadecimais de 7 Segmentos.

Endereços FFFEh a FFF3h: permitem escrever um dígito hexadecimal no mostrador correspondente.

Acelerómetro.

Endereços FFFBh a FFFD: permitem obter o valor da aceleração no eixo correspondente.

2.7 Interrupções

As interrupções indicam ao processador que existe um evento externo ao *pipeline* que necessita de atenção imediata. Existem 9 fontes possíveis de interrupções: o pressionar de cada um dos 7 botões de pressão, o pressionar de uma tecla no teclado PS/2 e o final da contagem do temporizador.

Quando é gerada uma interrupção, é testado se o bit E do registo de estado e o bit correspondente na máscara de interrupções têm valor 1, nesse caso, o processador interrompe a execução do programa e começa a executar a rotina de serviço dessa interrupção, que deverá estar no endereço calculado pela soma do endereço 7F00h com o valor do vetor de interrupção com um deslocamento de 4 bits para a esquerda. Por exemplo, a rotina de serviço às interrupções do temporizador, que tem um vetor de interrupção com valor Fh, deverá ser colocada no endereço 7FF0h (7F00h + F0h). Este deslocamento significa que cada rotina de interrupção pode utilizar um total de 16 posições de memória sem se sobrepor a outras rotinas de interrupção.

O valor dos vetores de interrupção e endereços das rotinas de serviço correspondentes estão indicados na Tabela 2.

Tabela 2. Vetores de interrupção e endereços das rotinas de serviço.

Origem	*Key0*	*Key1*	*Right*	*Up*	*Down*	*Left*	*Select*	Teclado	Temporizador
Vetor	0h	1h	2h	3h	4h	5h	6h	7h	Fh
Endereço	7F00h	7F10h	7F20h	7F30h	7F40h	7F50h	7F60h	7F70h	7FF0h

3 Assemblador

3.1 Evocação

Para evocar o assemblador e simulador do processador P4 basta executar o ficheiro p4 (Linux), ou p4.exe (Windows). No caso da versão online, basta aceder ao URL (indicado na página da cadeira) utilizando um navegador atualizado.

3.2 Conjunto de Instruções.

As instruções assembly suportadas pelo processador P4 estão apresentadas na Tabela 3.

Tabela 3. Conjunto de instruções do processador P4.

Aritméticas	Lógicas	Deslocamento	Controlo	Transferência	Genéricas
`NEG`	`COM`	`SHR`	`BR`	`MOV`	`NOP`
`INC`	`AND`	`SHL`	`BR.`*`cond`*	`LOAD`	`ENI`
`DEC`	`OR`	`SHRA`	`JMP`	`STOR`	`DSI`
`ADD`	`XOR`	`SHLA`	`JMP.`*`cond`*	`MVI`	`STC`
`ADDC`	`TEST`	`ROR`	`JAL`	`MVIH`	`CLC`
`SUB`		`ROL`	`JAL.`*`cond`*	`MVIL`	`CMC`
`SUBB`		`RORC`	`RTI`		
`CMP`		`ROLC`	`INT`		

As instruções BR, JMP e JAL podem ser seguidas de um código que indica uma condição necessária para a execução do salto.

3.3 Constantes

O tamanho máximo das constantes é de 16 bits, ou seja, valores compreendidos entre 0 e 65535 quando interpretadas como números sem sinal, ou entre -32768 e 32767 quando interpretadas como números em complemento para 2.

É possível especificar, no código *assembly*, constantes em decimal, hexadecimal, binário, octal e como caracteres:

– **Decimal:** valor constituído pelos algarismos de 0 a 9, opcionalmente seguido do caráter `d`. São válidos os valores entre `-32768` e `65535`.

- **Hexadecimal:** valor constituído pelos algarismos de 0 a 9 e pelas letras de A a F, seguido do caráter `h`. São válidos os valores entre `-8000h` e `FFFFh`.
- **Binário:** valor constituído pelos algarismos 0 e 1 seguido do caráter `b`. São válidos os valores entre `-1000000000000000b` e `1111111111111111b`.
- **Octal:** valor constituído pelos algarismos de 0 a 7 seguido do caráter `o`. São válidos os valores entre `-100000o` e `177777o`.
- **Caráter:** um caráter entre plicas. Esta constante tem como valor o código do caráter na codificação de caracteres ASCII estendido *Code Page 437*, ou Unicode, caso essa codificação tenha sido ativada através da diretiva `OPT`.

Por norma, as constantes são definidas no início do programa *assembly*, sendo-lhes associada uma etiqueta, o que permite tornar o código mais legível e facilita a atualização dessas constantes posteriormente.

3.4 Modos de Endereçamento

O P4 possui três modos de endereçamento: endereçamento por registo, endereçamento imediato e endereçamento indireto por registo.

Endereçamento por registo.
É o único modo possível para as instruções aritméticas, lógicas e de deslocamento. O operando corresponde ao registo indicado. Por exemplo: `ADD R1, R2, R3`.

Endereçamento imediato.
Este modo é utilizado em instruções de carregamento de constantes em registos, instruções de salto relativo e na instrução `INT`. O operando corresponde à constante indicada. Por exemplo: `BR 127`.

Endereçamento indireto por registo.
Este modo é utilizado pelas instruções `LOAD` e `STOR`. O operando corresponde à posição de memória cujo endereço é dado pelo valor do registo indicado. Por exemplo: `LOAD R1, M[R2]`.

3.5 Etiquetas

As instruções podem ser precedidas por "`<etiqueta>:`", que associa `<etiqueta>` à posição de memória onde irá ficar aquela instrução. Com isto, `<etiqueta>` pode ser usada como argumento em instruções de salto, o que permite aliviar o programador da tarefa de calcular o endereço ou o deslocamento necessário para realizar o salto. Por exemplo, se tivermos a instrução:

```
ReadLoop: LOAD R1, M[R2]
```

É possível realizar um salto absoluto, especificando a etiqueta, ao invés de um registo com o endereço da instrução:

```
JMP ReadLoop
```

O assemblador gera o código para carregar previamente o endereço de destino, `ReadLoop`, no registo R7 antes de executar a instrução `JMP R7`. Isto tem como consequência a perda do conteúdo do registo R7.

Também é possível utilizar etiquetas nos saltos relativos. O assemblador calcula automaticamente o deslocamento e usa-o na instrução de salto. Os saltos relativos estão limitados a um máximo de 128 posições para trás (PC-128) e 127 posições para a frente (PC+127) no programa.

As etiquetas podem ainda ser iniciadas pelo caráter "`.`", tornando-as em etiquetas locais, o que significa que ficam associadas à última etiqueta global definida. Desta forma, é possível haver múltiplas etiquetas locais com o mesmo nome, desde que associadas a diferentes etiquetas globais. É também possível referir uma etiqueta local associada a uma etiqueta global diferente da última etiqueta global definida, concatenando-se a etiqueta global com a etiqueta local. Por exemplo:

```
Rotina1:    ; código
.Loop       ; código
            JMP.Z   .Loop
Rotina2:    ; código
.Loop       ; código
            JMP.Z   .Loop
Rotina3:    ; código
            JMP.Z   Rotina1.Loop
```

Neste fragmento de código, ambas as instruções `JMP.Z .Loop` saltam para a instrução que as precedem, isto porque a etiqueta `.Loop` utilizada nestas instruções está associada à etiqueta global anterior.

Na realidade, a primeira definição de `.Loop` define o símbolo `Rotina1.Loop`, enquanto a segunda definição define o símbolo `Rotina2.Loop`. Desta forma, a instrução `JMP.Z Rotina1.Loop` salta para a instrução correspondente à etiqueta `.Loop` associada à etiqueta global `Rotina1`.

3.6 Comentários

Os comentários começam com o caráter "`;`". Todo o texto até ao final da linha é considerado comentário.

3.7 Diretivas

Além das instruções indicadas anteriormente, o assemblador reconhece um conjunto de diretivas que não se traduzem em instruções para o processador mas permitem associar símbolos a constantes, reservar espaço em memória, inicializar posições da memória e indicar a posição na memória onde deverão ficar as instruções. Nesta secção estão descritas as diretivas suportadas pelo assemblador do P4.

ORIG

Formato: `ORIG <endereço>`

Descrição: Define a primeira posição de memória em que um bloco de programa ou dados é carregado em memória, podendo ser usado múltiplas vezes de forma a definir blocos em diferentes zonas de memória. Não pode haver zonas sobrepostas.

EQU

Formato: `<símbolo> EQU <const>`

Descrição: Associa o valor `<const>` ao símbolo `<símbolo>`.

WORD

Formato: `<etiqueta> WORD <const>`

Descrição: Reserva uma posição de memória e inicializa-a com o valor `<const>`, associando o endereço dessa posição de memória ao nome `<etiqueta>`.

STR

Formato: `<etiqueta> STR '<texto>'|<const>(,'<texto>'|<const>)*`

Descrição: Reserva posições consecutivas de memória e inicializa-as com o valor de cada caráter de `<texto>` ou com o valor da constante `<const>`. É possível usar um número arbitrário de operandos, separando-os por vírgula, que serão avaliados de forma sequencial da esquerda para a direita. O nome `<etiqueta>` é associado à primeira posição de memória. Dois caracteres `'` consecutivos em `<texto>` são interpretados como um único caráter `'`.

TAB

Formato: `<etiqueta> TAB <const>`

Descrição: Reserva `<const>` posições consecutivas de memória sem as inicializar. O nome `<etiqueta>` é associado à primeira posição de memória.

OPT

Formato: `OPT <opção>`

Descrição: Permite configurar a assemblagem ativando a opção indicada pelo campo `<opção>`. As opções possíveis são:

- `ENABLE_DELAY_SLOTS`: ativa a utilização das *delay slots*. A primeira instrução *assembly* após cada instrução de salto é executada independentemente de o salto se realizar ou não.
- `DISABLE_DELAY_SLOTS`: desativa a utilização das *delay slots*. Esta opção faz com que o assemblador preencha automaticamente as *delay slots* com instruções `NOP`.
- `ASCII`: ativa a utilização da codificação de caracteres ASCII estendido CP437 no assemblador. Os caracteres em `'<texto>'` serão tratados como ASCII estendido CP437. Por exemplo, `'É'` será interpretado como a constante 0090h.
- `UNICODE`: ativa a utilização da codificação de caracteres Unicode no assemblador. Os caracteres em `'<texto>'` serão tratados como Unicode. Por exemplo, `'É'` será interpretado como a constante 00C9h. Esta opção não altera a codificação utilizada pelos periféricos.

As opções `DISABLE_DELAY_SLOTS` e `ASCII` encontram-se ativadas por omissão.

Os símbolos e etiquetas definidos pelas diretivas `EQU`, `WORD`, `STR` e `TAB` podem ser compostos por caracteres alfanuméricos (A-Z, a-z e 0-9) mais o caráter _, e o primeiro caráter não pode ser um algarismo.

3.8 Instruções Assembly

As instruções Assembly do processador P4 são apresentadas nesta secção de forma detalhada. Salvo indicação em contrário, os operandos são registos.

ADD *Flags*: ZCNO

Formato: `ADD op1, op2, op3`

Ação: $op1 \leftarrow op2 + op3$

Descrição: Soma op3 a op2 e guarda o resultado em op1.

ADDC *Flags*: ZCNO

Formato: `ADDC op1, op2, op3`

Ação: $op1 \leftarrow op2 + op3 + C$

Descrição: Soma op3 a op2 e ao valor do bit de estado transporte, guardando o resultado em op1.

AND *Flags*: ZN

Formato: `AND op1, op2, op3`

Ação: $op1 \leftarrow op2 \wedge op3$

Descrição: Faz a disjunção lógica bit a bit entre op2 a op3 e guarda o resultado em op1.

BR *Flags*: Nenhuma

Formato: `BR <deslocamento>`

Ação: $PC \leftarrow PC + deslocamento$

Descrição: Salto relativo incondicional. A nova posição é determinada em relação à posição da instrução de salto somando-se o valor de <deslocamento>. O valor tem de estar compreendido entre -128 e 127. É possível utilizar uma etiqueta de forma a realizar o salto para a instrução precedida por essa etiqueta.

BR.*cond* *Flags*: Nenhuma

Formato: `BR.`*`cond`* `<deslocamento>`

Ação: $PC \leftarrow PC + deslocamento$

Descrição: Salto relativo condicional. Funciona de modo similar ao salto relativo incondicional com a diferença de que caso a condição não se verifique esta instrução não surte qualquer efeito. As condições possíveis são:

Condição	Zero	Negativo	Positivo	Transporte	Excesso
Verdadeiro	`BR.Z`	`BR.N`	`BR.P`	`BR.C`	`BR.O`
Falso	`BR.NZ`	`BR.NN`	`BR.NP`	`BR.NC`	`BR.NO`

CLC *Flags*: C

Formato: `CLC`

Ação: $C \leftarrow 0$

Descrição: *Clear Carry*. Coloca o bit de estado transporte a 0.

CMC *Flags*: C

Formato: `CMC`

Ação: $C \leftarrow \overline{C}$

Descrição: *Complement Carry*. Complementa o valor do bit de estado de transporte.

CMP *Flags*: ZCNO

Formato: `CMP op1, op2`

Ação: $op1 - op2$

Descrição: Compara os operandos op1 e op2, atualizando os bits de estado. É equivalente à instrução `SUB R0, op1, op2`. É normalmente utilizada antes de saltos condicionais.

COM *Flags*: ZN

Formato: `COM op`

Ação: $op \leftarrow \overline{op}$

Descrição: Faz o complemento bit a bit de op.

DEC *Flags*: ZCNO

Formato: `DEC op`

Ação: $op \leftarrow op - 1$

Descrição: Decrementa op em uma unidade.

DSI *Flags*: E

Formato: `DSI`

Ação: $E \leftarrow 0$

Descrição: *Disable Interrupts*. Coloca o bit de estado E a 0, inibindo o tratamento de interrupções.

ENI *Flags*: E

Formato: `ENI`

Ação: $E \leftarrow 1$

Descrição: *Enable Interrupts*. Coloca o bit de estado E a 1, ativando o tratamento de interrupções.

INC *Flags*: ZCNO

Formato: `INC`

Ação: $op \leftarrow op + 1$

Descrição: Incrementa op em uma unidade.

INT *Flags*: ZCNOE

Formato: `INT <constante>`

Ação: $PC \leftarrow 7F00h + constante, \quad PC_E \leftarrow PC, \quad RE_E \leftarrow RE$

Descrição: Gera uma interrupção independentemente do valor do bit de estado E. O endereço da rotina de tratamento da interrupção é calculado como a soma entre 7F00h e <constante>, que deverá estar compreendida entre 0 e 255. Guarda o valor do *Program Counter* e o Registo de Estado para ser utilizado mais tarde pela instrução `RTI`.

JAL *Flags*: Nenhuma

Formato: `JAL <endereço>`

Ação: $PC \leftarrow$ ***endereço***, $\quad R7 \leftarrow PC + 1$

Descrição: Salto absoluto incondicional. A nova posição é determinada pelo valor de <endereço>. A posição atual é guardada em R7. É possível utilizar uma etiqueta de forma a realizar o salto para a instrução precedida por essa etiqueta, ou um registo, caso em que salta para o endereço contido nesse registo.

JAL.*cond* *Flags*: Nenhuma

Formato: `JAL.`*`cond`* `<endereço>`

Ação: $PC \leftarrow$ ***endereço***, $\quad R7 \leftarrow PC + 1$

Descrição: Salto absoluto condicional. Funciona de modo similar ao salto absoluto incondicional com a diferença de que caso a condição não se verifique esta instrução não surte qualquer efeito. As condições possíveis são:

Condição	Zero	Negativo	Positivo	Transporte	Excesso
Verdadeiro	`JAL.Z`	`JAL.N`	`JAL.P`	`JAL.C`	`JAL.O`
Falso	`JAL.NZ`	`JAL.NN`	`JAL.NP`	`JAL.NC`	`JAL.NO`

JMP *Flags*: Nenhuma

Formato: `JMP <endereço>`

Ação: $PC \leftarrow$ ***endereço***

Descrição: Salto absoluto incondicional. A nova posição é determinada pelo valor de <endereço>. É possível utilizar uma etiqueta de forma a realizar o salto para a instrução precedida por essa etiqueta, ou um registo, caso em que salta para o endereço contido nesse registo.

JMP.*cond* *Flags*: Nenhuma

Formato: `JMP.cond <endereço>`

Ação: $PC \leftarrow$ *endereço*

Descrição: Salto absoluto condicional. Funciona de modo similar ao salto absoluto incondicional com a diferença de que caso a condição não se verifique esta instrução não surte qualquer efeito. As condições possíveis são:

Condição	Zero	Negativo	Positivo	Transporte	Excesso
Verdadeiro	`JMP.Z`	`JMP.N`	`JMP.P`	`JMP.C`	`JMP.O`
Falso	`JMP.NZ`	`JMP.NN`	`JMP.NP`	`JMP.NC`	`JMP.NO`

LOAD *Flags*: Nenhuma

Formato: `LOAD op1, op2`

Ação: $op1 \leftarrow op2$

Descrição: Copia o valor de op2 para op1. op2 deverá ser uma posição de memória endereçada por registo indireto `M[Rx]`, em que o endereço dessa posição é o conteúdo do registo `Rx`.

MOV *Flags*: Nenhuma

Formato: `MOV op1, op2`

Ação: $op1 \leftarrow op2$

Descrição: Copia o valor de op2 para op1.

MVI *Flags*: Nenhuma

Formato: `MVI op1, op2`

Ação: $op1 \leftarrow op2$

Descrição: Copia o valor de op2 para op1. op2 deverá ser uma constante compreendida entre -32768 e 65535. Para valores superiores a 255 ou inferiores a -128 esta instrução é automaticamente convertida num par de instruções MVIH e MVIL.

MVIH *Flags*: Nenhuma

Formato: `MVI op1, op2`

Ação: $op1 \leftarrow (op1 \wedge 00FFh) \vee (op2 << 8)$

Descrição: Copia o octeto de maior peso de op2 para o octeto de maior peso de op1. op2 deverá ser uma constante compreendida entre -128 e 255.

MVIL *Flags*: Nenhuma

Formato: `MVI op1, op2`

Ação: $op1 \leftarrow (op1 \wedge FF00h) \vee op2$

Descrição: Copia o octeto de menor peso de op2 para o octeto de menor peso de op1. op2 deverá ser uma constante compreendida entre -128 e 255.

NEG *Flags*: ZCNO

Formato: `NEG op`

Ação: $op \leftarrow -op$

Descrição: Inverte o sinal de op. Equivalente à instrução `SUB op, R0, op`.

NOP *Flags*: Nenhuma

Formato: `NOP`

Ação: $\emptyset$

Descrição: *No operation*. Não tem qualquer efeito.

OR *Flags*: ZN

Formato: `OR op1, op2, op3`

Ação: $op1 \leftarrow op2 \vee op3$

Descrição: Faz a conjunção lógica bit a bit entre op2 a op3 e guarda o resultado em op1.

ROL *Flags*: ZCN

Formato: `ROL op`

Ação: $op \leftarrow (op << 1) \vee (op >> 15)$

Descrição: *Rotate Left*. Faz a rotação à esquerda dos bits de op. Similar ao deslocamento à esquerda com a diferença de que o bit mais à esquerda passa a ser o bit mais à direita.

ROLC *Flags*: ZCN

Formato: `ROLC op`

Ação: $op \leftarrow (op << 1) \vee C, \quad C \leftarrow op >> 15$

Descrição: *Rotate Left with Carry*. Similar à rotação à esquerda com a diferença de que o bit de estado transporte passa a ser o bit mais à direita e o bit mais à esquerda é colocado no bit de estado transporte.

ROR *Flags*: ZCN

Formato: `ROR op`

Ação: $op \leftarrow (op >> 1) \vee ((op \wedge 1) << 15)$

Descrição: *Rotate Right*. Faz a rotação à direita dos bits de op. Similar ao deslocamento à direita com a diferença de que o bit mais à direita passa a ser o bit mais à esquerda.

RORC *Flags*: ZCN

Formato: `ROLC op`

Ação: $op \leftarrow (op >> 1) \vee (C << 15), \quad C \leftarrow op \wedge 1$

Descrição: *Rotate Right with Carry*. Similar à rotação à direita com a diferença de que o bit de estado transporte passa a ser o bit mais à esquerda e o bit mais à direita é colocado no bit de estado transporte.

RTI *Flags*: ZCNOE

Formato: `RTI`

Ação: $PC \leftarrow PC_E, \quad RE \leftarrow RE_E$

Descrição: *Return from Interrupt*. Executa um salto para a instrução que iria ser executada antes do tratamento da interrupção e repõe o registo de estado com o valor do estado antes do tratamento da interrupção.

SHL *Flags*: ZCN

Formato: `SHL op`

Ação: $op \leftarrow op << 1, \quad C \leftarrow op >> 15$

Descrição: *Shift Left*. Faz o deslocamento à esquerda dos bits de op. É inserido um 0 na posição mais à direita e o bit mais à esquerda é colocado no bit de estado transporte.

SHLA *Flags*: ZCNO

Formato: `SHLA op`

Ação: $op \leftarrow op << 1, \quad C \leftarrow op >> 15$

Descrição: *Shift Left Arithmetic*. Similar ao deslocamento à esquerda com a diferença de que os bits de estado correspondente às operações aritméticas são atualizados.

SHR *Flags*: ZCN

Formato: `SHR op`

Ação: $op \leftarrow op >> 1, \quad C \leftarrow op \wedge 1$

Descrição: *Shift Right*. Faz o deslocamento à direita dos bits de op. É inserido um 0 na posição mais à esquerda e o bit mais à direita é colocado no bit de estado transporte.

SHRA *Flags*: ZCNO

Formato: `SHRA op`

Ação: $op \leftarrow op >> 1, \quad C \leftarrow op \wedge 1$

Descrição: *Shift Right Arithmetic*. Similar ao deslocamento à direita com a diferença de que os bits de estado correspondente às operações aritméticas são atualizados.

STC *Flags*: C

Formato: `STC`

Ação: $C \leftarrow 1$

Descrição: *Set Carry*. Coloca o bit de estado transporte a 1.

STOR *Flags*: Nenhuma

Formato: `STOR op1, op2`

Ação: $op1 \leftarrow op2$

Descrição: Copia o valor de op2 para op1. op1 deverá ser uma posição de memória endereçada por registo indireto `M[Rx]`, em que o endereço dessa posição é o conteúdo do registo `Rx`.

SUB *Flags*: ZCNO

Formato: `SUB op1, op2, op3`

Ação: $op1 \leftarrow op2 - op3$

Descrição: Subtrai op3 a op2 e guarda o resultado em op1.

SUBB *Flags*: ZCNO

Formato: `SUBB op1, op2, op3`

Ação: $op1 \leftarrow op2 - op3 - (1 - C)$

Descrição: Subtrai op3 e o complemento do valor do bit de estado transporte a op2, guardando o resultado em op1.

TEST *Flags*: ZN

Formato: `TEST op1, op2`

Ação: $op1 \wedge op2$

Descrição: Testa os bits dos operandos op1 e op2, atualizando os bits de estado. É equivalente à instrução `AND R0, op1, op2`. É normalmente utilizada antes de saltos condicionais.

XOR *Flags*: ZN

Formato: `XOR op1, op2, op3`

Ação: $op1 \leftarrow op2 \oplus op3$

Descrição: Faz a conjunção lógica exclusiva bit a bit entre op2 e op3 e guarda o resultado em op1.

4 Simulador

4.1 Evocação

O simulador está integrado no assemblador. Basta evocar o assemblador como indicado anteriormente.

4.2 Ambiente

O simulador apresenta várias secções:

Botões de controlo. Existem 3 botões para controlar o simulador. Iniciar/Parar (Start/Stop), Passo (Step) e Reiniciar (Reset).

O botão Iniciar coloca o simulador em funcionamento e irá correr o programa assembly carregado no assemblador.

O botão Parar irá colocar o processador em pausa. Ao clicar novamente em Iniciar o simulador retoma a partir do estado em que a simulação parou.

O botão Reiniciar simula um Reset ao processador. Os registos e periféricos são reinicializados. O conteúdo da memória mantém-se.

Indicação de períodos de relógio. Número de períodos de relógio simulados desde o início da simulação ou da última reinicialização.

Velocidade da simulação. Velocidade da simulação em termos de ciclos de relógio por segundo.

Controlo da velocidade de simulação. Este controlo permite definir a velocidade máxima da simulação.

Registos e Instrução atual. Este painel indica o valor do vários registos: PC, R1 a R7 e registo de estado. O valor RI corresponde ao valor da instrução a ser executada.

Ao clicar neste painel é possível alternar entre a representação hexadecimal, decimal sem sinal, decimal com sinal (a partir do complemento para 2) e binária.

Periféricos. Os periféricos da placa do P4 estão representados no simulador. É possível ver o texto que seria mostrado no LCD e na janela de texto, o valor que seria mostrado nos mostradores aHexadecimais de 7 Segmentos e os LED. Existem controlos para definir o estado dos interruptores, o valor lido pelo acelerómetro, o premir dos botões de pressão, e o texto introduzido no teclado.

Conteúdo da Memória. Os painéis de visualização da memória permitem ver o conteúdo da memória de programa e da memória de dados.

Por questões de eficiência estes painéis mostram um número limitado de posições de memória. É possível definir o endereço da primeira posição de memória a ser mostrada através de uma caixa de texto, ou de dois *sliders* que definem o byte mais e menos significativo do endereço. Ao deslocar a roda do rato sobre o painel o endereço é automaticamente incrementado ou decrementado.

4.3 Depuração

Para ajudar na depuração do programa Assembly, é possível criar pontos de paragem. Para tal, clica-se sobre o número da linha no editor de Assembly, aparecendo um sinal para indicar a existência de um ponto de paragem.

Quando o simulador estiver a correr (i.e. após clicar Iniciar), a simulação irá automaticamente parar imediatamente antes de executar uma instrução que tenha um ponto de paragem definido.

É possível então analisar o estado da simulação imediatamente antes da execução dessa instrução, seguidamente pode carregar-se em Passo para observar o resultado de executar essa instrução, ou Iniciar para continuar a simulação até ao próximo ponto de paragem.

Quando a simulação está parada ou a ser executada a baixa velocidade, a linha de código correspondente à próxima instrução a ser executada aparece sombreada.

5 Dispositivos de Entrada e Saída

A Tabela 4 mostra o mapeamento em memória das entradas e saídas suportadas pelo simulador do P4.

Tabela 4. Mapeamento em memória dos dispositivos de entrada e saída.

Endereço	Dispositivo	Função	Leitura	Escrita
FFFF	Terminal	Ler última tecla premida	X	
FFFE	Terminal	Escrever caráter		X
FFFD	Terminal	Testar se houve tecla premida	X	
FFFC	Terminal	Posicionar cursor		X
FFFB	Terminal	Definir cor		X
FFFA	Máscara de Interrupções	Ler ou definir vetores de interrupção ativados	X	X
FFF9	Interruptores	Ler valor dos interruptores	X	
FFF8	LED	Definir estado ligado/desligado dos LED		X
FFF7	Temporizador	Iniciar/parar ou ler estado	X	X
FFF6	Temporizador	Ler ou definir valor da contagem	X	X
FFF5	Mostrador LCD	Escrever caráter		X
FFF4	Mostrador LCD	Posicionar cursor		X
FFF3	Mostrador 3	Escrever valor no mostrador 3		X
FFF2	Mostrador 2	Escrever valor no mostrador 2		X
FFF1	Mostrador 1	Escrever valor no mostrador 1		X
FFF0	Mostrador 0	Escrever valor no mostrador 0		X
FFEF	Mostrador 5	Escrever valor no mostrador 5		X
FFEE	Mostrador 4	Escrever valor no mostrador 4		X
FFED	Acelerómetro Z	Ler aceleração no eixo Z	X	
FFEC	Acelerómetro Y	Ler aceleração no eixo Y	X	
FFEB	Acelerómetro X	Ler aceleração no eixo X	X	

Nesta secção será explicado como utilizar os portos mapeados em memória para controlar os dispositivos ou para obter valores das entradas.

5.1 Terminal.

Ao fazer a leitura do porto de leitura, FFFFh, obtem-se o valor ASCII da última tecla premida, ou zero caso nenhuma tecla tenha sido premida desde a última leitura.

A escrita para o porto de escrita, FFFEh, resulta na escrita de um caráter na janela de texto na posição atual do cursor, fazendo-o deslocar-se para a posição seguinte, da esquerda para a direita e de cima para baixo, voltando à posição inicial após a última posição.

O caráter a ser escrito é definido pelos 8 bits menos significativos do valor escrito para este porto e ao caráter correspondente em ASCII estendido (*Code Page 437*).

A leitura do porto de estado, FFFDh, permite saber se houve alguma tecla premida desde a última leitura do porto de leitura. Caso tenha havido, o valor lido será 1, caso contrário, será 0.

A escrita para o porto de controlo, FFFCh, define a posição do cursor. Os 8 bits mais significativos do valor escrito determinam a linha, enquanto os 8 bits menos significativos determinam a coluna. O valor da linha deverá estar entre 0 e 44, enquanto que o valor da coluna deverá estar entre 0 e 79.

A escrita para o porto de cor, FFFBh, define a cor do caráter e do fundo nas escritas subsequentes. Os 8 bits mais significativos do valor escrito determinam a cor do fundo, enquanto os 8 bits menos significativos determinam a cor do caráter. Os bits de cor estão codificados como RRRGGGBB, em que RRR corresponde ao valor da intensidade da componente vermelha, e similarmente, GGG e BB para as componentes verde e azul, respetivamente. Por exemplo, EC03h (111011000000011b), irá corresponder a letras azuis sobre um fundo laranja.

5.2 Máscara de Interrupções.

A escrita para o porto FFFAh permite ativar ou desativar cada um dos primeiros 16 vetores de interrupção. O vetor estará ativo quando o bit correspondente no valor escrito para este porto tiver valor 1. O bit menos significativo corresponde ao primeiro vetor de interrupção, enquanto que o mais significativo corresponde ao 16.º vetor de interrupção. Por exemplo, a escrita do valor 8000h para este porto significará que apenas o 16.º vetor (interrupções causadas pelo temporizador) ficará ativo.

5.3 Interruptores.

Ao fazer a leitura do porto FFF9h obtém-se o estado dos interruptores. Quando o interruptor estiver para cima, o bit correspondente terá valor 1, quando estiver para baixo, o bit correspondente terá valor 0. O interruptor mais à direita corresponde ao bit menos significativo.

5.4 LED.

A escrita para o porto FFF8h define o estado dos LED. O LED irá ficar aceso quando o bit correspondente tiver valor 1, e ficará apagado quando o bit correspondente tiver valor 0. O led mais à direita corresponde ao bit menos significativo.

5.5 Temporizador.

A escrita para o porto FFF7h permite iniciar ou parar o temporizador. Quando o bit menos significativo do valor escrito for 1, o temporizador irá iniciar, caso seja 0, o temporizador pára. A leitura deste porto devolve o valor 1 caso o temporizador esteja em contagem e 0 caso contrário.

A escrita para o porto FFF6h irá definir a duração da contagem. A duração será determinada pelo valor escrito para este porto em décimas de segundo, i.e., cada unidade corresponde a 100 ms. A leitura deste porto devolve o valor atual da contagem, ou seja, as décimas de segundo decorridas desde o início da contagem.

5.6 Ecrã LCD.

A escrita para o porto de escrita, FFF5h, resulta na escrita de um caráter na janela de texto na posição atual do cursor, fazendo-o deslocar-se para a posição seguinte, da esquerda para a direita e de cima para baixo, voltando à posição inicial após a última posição.

O caráter a ser escrito é definido pelos 8 bits menos significativos do valor escrito para este porto e a sua correspondência em ASCII. Para valores superiores a 127 o caráter escrito está definido no *datasheet* do ecrã LCD Hitachi modelo HD44780U, ROM Code A00.

A escrita para o porto de controlo, FFF4h, permite definir a posição do cursor, o estado de ligado/-desligado do ecrã LCD, bem como limpar o ecrã LCD. Essas ações são determinadas pelos bits do valor escrito, da seguinte forma:

Bit	Ação
15	Liga (1) ou desliga (0) o ecrã.
5	Limpa o ecrã (1).
4	Posiciona o cursor na primeira (0) ou segunda linha (1).
3 a 0	Posiciona o cursor na coluna especificada (0 a 15).

5.7 Mostradores Hexadecimais de 7 Segmentos.

A escrita nos portos FFEEh a FFF3h permite definir o valor hexadecimal, de 0h a Fh, mostrado pelo mostrador correspondente. O valor mostrado é determinado pelos 4 bits menos significativos do valor escrito para este porto. O mostrador 0 situa-se mais à direita, o mostrador 1 situa-se à esquerda do mostrador 0, e assim sucessivamente.

5.8 Acelerómetro.

A leitura dos portos FFEBh a FFED permite obter o valor da aceleração no eixo correspondente. O valor de 16 bits deverá ser interpretado em complemento para 2. O valor 7FFFh corresponde à aceleração máxima de 2 g, enquanto que o valor 00FFh corresponde a 1 g.