

Software Básico

Primeiro Trabalho Data Limite: 01/06/2022

Prof. Pedro Carlos da Silva Lara

Centro Federal de Educação Tecnológica Celso Suckow da Fonseca

Processador SP1 (Simple Processor 1)

1 Informações Gerais

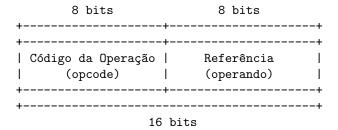
SP1 é um processador hipotético especificado com fim puramente acadêmico. Ele possui três registradores de 8 bits (1 byte) cada um: acc (acumulador) e stat (status) e pc (program counter). O ISA do processador SP1 é composto por 16 instruções. O formato das instruções são fixas, neste caso 16 bits (8 bits para o opcode e 8 bits para o operando). O registrador stat mantém informações de carry, overflow e zero do registrador acc. O objetivo do trabalho é desenvolver um emulador, em linguagem C, para o processador SP1. Os detalhes do SP1 serão vistos nas Seções seguintes.

2 Registradores

O registrador acumulador (acc) recebe todos os resultados das operações lógicas e aritméticas. Sendo um registrador essencial em arquiteturas de 1 endereço (1 operando) baseada em acumulador. O registrador pc (program counter) contém a posição de memória (endereço) onde se deve buscar a próxima instrução.

3 Formato das Instruções

O SP1 suporta instruções fixas de 16 bits como especificado abaixo:



O campo código da operação ou opcode identifica unicamente a instrução que será realizada. Desta forma, cada instrução possui um opcode associado. O campo referência identifica o valor ou a localização dos dados que serão utilizados na instrução. Como o SP1 possui apenas 16 instruções, seria necessário apenas 4 bits para a representação da operação. Para o campo Referência podemos utilizar dados com 8 bits. Assim, podemos referenciar (endereçar) até 28 bytes na memória. O trecho de código abaixo mostra como funciona o algoritmo para buscar uma instrução na memória:

4 Instruções

Instrução	opcode	Comentário
load	$0 (00000000_2)$	memória
load	1 (00000012)	valor
store	$2 (00000010_2)$	_
add	3 (000000112)	_
sub	4 (00000100 ₂)	_
mul	5 (00000101 ₂)	_
div	$6 (00000110_2)$	_
inc	7 (00000111 ₂)	_
dec	8 (000010002)	_
and	9 (00001001 ₂)	_
or	$10 (00001010_2)$	_
not	$11 (00001011_2)$	_
jmp	$12 (00001100_2)$	_
jz	$13 (00001101_2)$	_
jnz	$14 (00001110_2)$	_
hlt	$15 (00001111_2)$	fim do programa

4.1 LOAD

A instrução load opera de duas maneiras distintas: carrega um valor direto para o acumulador ou carrega um valor em **memória** para o acumulador. O trecho de código abaixo exibe as diferenças. Exemplo de código:

```
load 10 ; acc = 10. Carrega o valor 10 no acumulador
load $10 ; acc = mem[10]. Carrega o valor que tem endereço 10 no acumulador
```

Formato de Código : load \$addr

Código da Operação : 00

Função : Carrega o valor da posição de memória 'addr' em acc

Exemplo de instrução:

	opcode		addr	
İ	00000000	İ	00001010	İ
Τ.	load		\$10	

Formato de Código : load num

Código da Operação : 01

Função : Carrega o valor de 'num' em acc

Exemplo de instrução:

	opcode		addr	
•	00000001	•		•
+-		+-		+
	load		11	

4.2 STORE

Formato de Código : store \$addr

Código da Operação : 02

Função : Armazena o valor do acumulador na posição de memória definida por 'addr'

Exemplo de instrução:

```
opcode addr
+------+
| 00000010 | 10000111 |
+------+
store $135; mem[135] = acc
```

4.3 ADD

Operação de adição entre o acumulador e um valor em memória.

Formato de Código $\,$: add \$addr

Código da Operação : 03

Função : acc = acc + mem[addr]

Exemplo de instrução:

```
opcode addr
+------+
| 00000011 | 00100101 |
+-----+
add $37
```

4.4 SUB

Operação de subtração entre o acumulador e um valor em memória

Formato de Código : sub \$addr

Código da Operação : 04

Função : acc = acc - mem[addr]

Exemplo de instrução:

```
opcode addr
+-----+
| 00000100 | 11000010 |
+-----+
sub $194
```

4.5 MUL

Operação de multiplicação entre o acumulador e um valor em memória.

Formato de Código $\,:\,$ mul \$addr

Código da Operação : 05

Função : acc = acc * mem[addr]

Exemplo de instrução:

```
opcode addr
+-----+
| 00000101 | 00100100 |
+-----+
mul $36
```

4.6 DIV

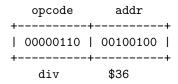
Operação de divisão entre o acumulador e um valor em memória.

Formato de Código : div \$addr

Código da Operação : 06

Função : acc = acc / mem[addr]

Exemplo de instrução:



4.7 INC

Incrementa em uma unidade o acc.

Formato de Código : inc Código da Operação : 07

Função : acc = acc + 1

Exemplo de instrução:

```
opcode addr
+------+
| 00000111 | 00000000 |
+-----+
inc
```

4.8 DEC

Decrementa em uma unidade o acc.

Formato de Código : dec Código da Opervação : 08

Função : acc = acc - 1

Exemplo de instrução:

```
opcode addr
+-----+
| 00001000 | 00000000 |
+----+
dec
```

4.9 AND

Operação de E-lógico entre o acumulador e um valor em memória.

Formato de Código : and \$addr

Código da Operação: 09

Função : acc = acc & mem[addr]

Exemplo de instrução:

```
opcode addr
+-----+
| 00001001 | 00000100 |
+-----+
and $4
```

4.10 OR

Operação de **OU-lógico** entre o acumulador e um valor em memória.

Formato de Código $\,:\,$ or \$addr

Código da Operação : 10

Função : acc = acc | mem[addr]

Exemplo de instrução:

opcode			addr		
+-		+-		+	
1	00001010	1	00001100	I	
+-		+-		+	
	or		\$12		

4.11 NOT

Operação de negação de todos os bits do acumulador: executa o complemento a 2 no acumulador.

Formato de Código : not Código da Operação : 11

Função : acc = ~acc

Exemplo de instrução:

4.12 JMP

Desvio incondicional para o label.

Formato de Código : jmp #label

Código da Operação : 12

Função : desvia a execução para

a definição de 'label'

Exemplo de instrução:

	opcode		addr	
l	00001100	İ	00001101	İ
T-	 jmp		13	

4.13 JZ

Desvia caso o resgistrador acc seja igual a 0. Caso contrário, prossiga no código. Desta forma, o registrador pc será modificado se o acc for igual a 0.

Formato de Código : jnz #label

Código da Operação : 13

Função : Se acc == 0, desvia a execução para

a definição de 'label'

Exemplo de instrução:

```
opcode ref.

+------+

| 00001101 | 01001110 |

+------+

jz 78
```

4.14 JNZ

Desvia caso o registrador acc seja diferente de 0. Caso contrário, prossiga no código. Desta forma, o registrador por será modificado se o acc for diferente de 0.

Formato de Código : jz #label

Código da Operação : 14

Função : Se acc != 0, desvia a execução para

a definição de 'label'

O assembler vas traduz cada *label* em uma posição do programa. **Exemplo de instrução:**

```
opcode ref.

+------+

| 00001110 | 00000111 |

+------+

jz 7
```

No exemplo acima, se no momento que foi executada a função jz o acumulador for zero program counter deverá ser alterado para 7.

4.15 HLT

Indica o fim do programa.

Formato de Código : hlt Código da Operação : 15

Função : Fim do programa

Exemplo de instrução:

```
opcode
+-----+
| 00001111 | 00000000 |
+-----+
hlt
```

5 Registrador STAT

STAT (8 bits)

+----+ | xxxxx0CZ | +----+

0 : Overflow
C : Carry
Z : Zero ACC
x : não utilizado

6 Exemplos

```
; soma acc = 10 + 20
load 20 ; acc = 20
```

store \$1 ; mem[1] = 20 load 10 ; acc = 10

add \$1 ; acc = acc + mem[1] hlt ; fim do programa

Programa executável:

O programa executável constitui apenas os valores. Se esse programa fosse visualizado por bytes em representação de base 16 (hexadecimal) teríamos o seguinte programa:

0x01 0x14 0x02 0x01 0x01 0x0A 0x03 0x01 0x0F 0x00

```
; Exemplo de loop
; acc = 10 + 9 + 8 + \dots + 1
load 0
store $1 ; $1 = 0
load 1
store $3 ; $3 = 1
load 10 ; acc = 10
#While:
    store $2; $2 = acc
   load $1; acc = $1
   add
         $2 ; acc = $2 + $1
    store $1 ; $1 = acc
   load $2; acc = $2
         $3 ; acc = acc - 1
    jnz #While
load $1
hlt
```