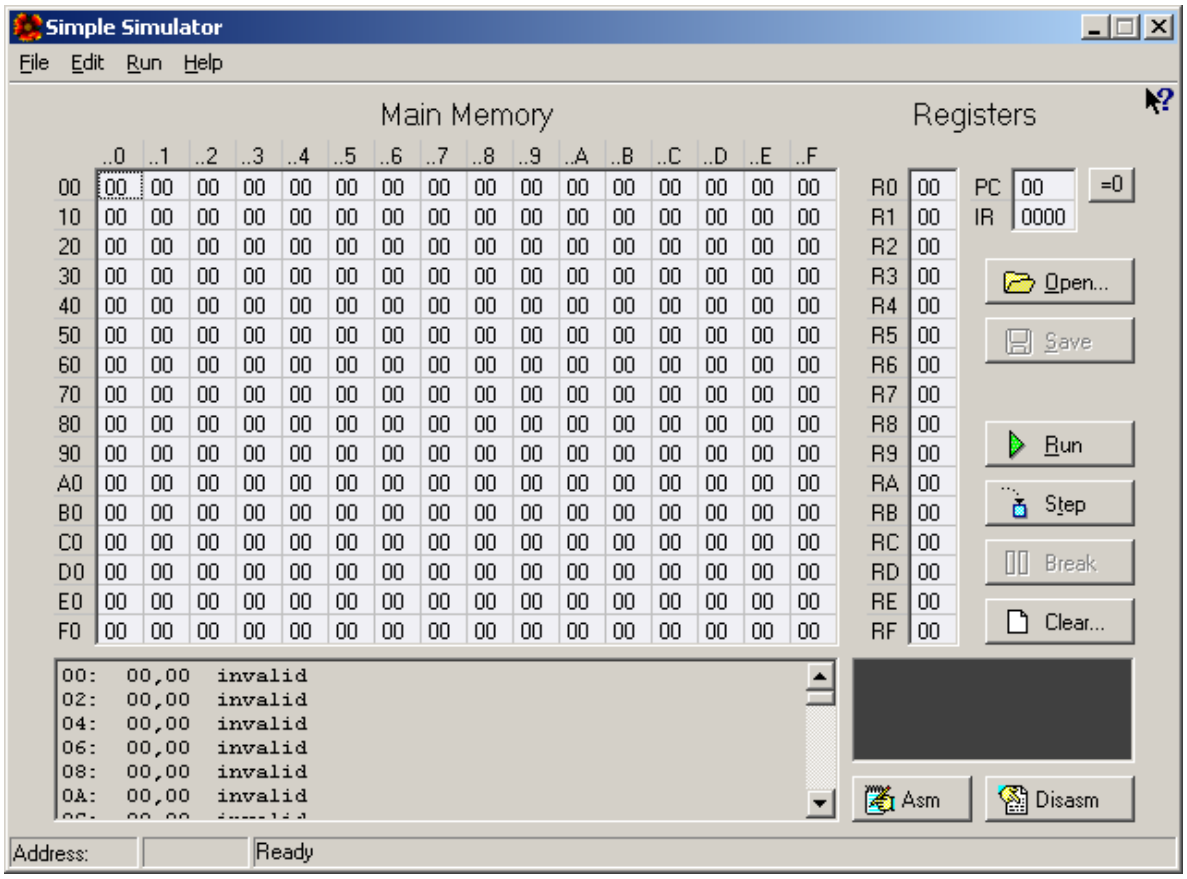
Utilizando o Simulador SimpSIM



Para utilizarmos o simulador é preciso criar um arquivo com extensão **.asm**, no qual devem ser descritas as instruções a serem executadas.
Cada linha pode ser composta por um rótulo, uma instrução ou um comentário.
Um rótulo é uma seqüência de caracteres (ou nome) que podemos usar para identificar uma sub-rotina (trecho de código).
Por exemplo, uma parte do código responsável por realizar a conversão de um número em complemento de 2 (de positivo para negativo) pode ser nomeada por **conversao:** .

Uma instrução pode ser definida por seu mnemônico seguida de seus operandos. Por exemplo, **addi R1, R2, R3**.

Um comentário pode ser feito usando qualquer seqüência de caracteres, desde que precedida por ponto-e-vírgula (;). Por exemplo:
; este é um comentário.

Os valores numéricos que podemos trabalhar no SimpSIM são os seguintes:

- **Número decimal** nas seguintes formas: -9, 10 ou 12**d** *(Nesse caso, o caractere **d** é opcional)*.
- **Número binário:** 101101**b**
- **Número hexadecimal: 0x**12AF ou **$**FAFF ou FA12**h**

Obs: não são permitidos espaços entre os números.

Os textos devem ser expressos entre aspas (“..”) ou apóstrofos (‘..’).

O editor utiliza-se de destaques (highlighting) na sintaxe:

Palavras reservadas: **`load, store, addi`**
Números: `-123, 0x10, 11001011b`
Strings: `“uma string”, ‘esta também é uma string’`
Comentários: `; este é um comentário`
Erros de sintaxe: `12A3, -0x12, 1+1`

Mnemônicos e a Combinação dos operandos

Conjunto de Dados em Bytes *(Data byte)*

```
db item_dado1, item_dado2, ...
```

Insere um conjunto de dados diretamente na memória principal. Cada item de dado (`item_dado1`) pode ser um número ou uma string.
Nesse caso, é possível especificar uma quantidade ilimitada de itens de dado.

Exemplo de uso:

```
db    1,4,9,16,25,36
db    “Hello World!”,0
```

Origem (*Origin*)

```
org endereço
```

O próximo código iniciará no endereço indicado nesta instrução. O operando `endereço` deve ser um número e não é permitida a sobreposição de diferentes fragmentos de código.

Exemplo de uso:

```
org 60h
load R0, 2   ; insere esta instrução no endereço $60
```

Carga imediata (*Immediate load*)

```
load reg, número
load reg, rótulo
```

Associa o valor imediato (`número` ou `endereço de um rótulo`) ao registrador `reg`.

Exemplo de uso:

```
load R4, 8
load R9, Conversao   ; Conversao é o rótulo de um trecho de código.
```

Carga direta (*Direct load*)

```
load reg, [endereco]
```

Associa o conteúdo de memória que está no `endereço` indicado. O `endereço` pode ser um número ou um rótulo.

Exemplo de uso:

```
load R4, [8]
load R9, [Conversao]
```

Carga indireta (*Indirect load*)

`load` `reg1, [reg2]`

Associa o conteúdo de memória cujo endereço está indicado no registrador `reg2` ao registrador `reg1`.

Exemplo de uso:

```
load R4, [R8]
```

Armazenamento direto (*Direct store*)

`store` `reg, [endereço]`

Armazena o valor do registrador `reg` em determinado `endereço` de memória. O `endereço` pode ser um número ou rótulo.

Exemplo de uso:

```
store R4, [8]
store R9, [Local]
```

Armazenamento indireto (*Indirect store*)

`store` `reg1, [reg2]`

Armazena o valor do registrador `reg1` no `endereço` de memória indicado no registrador `reg2`.

Exemplo de uso:

```
store R4, [R8]
```

Move

`move` `reg1, reg2`

Associa o valor do registrador `reg2` ao registrador `reg1`.

Exemplo de uso:

```
move R4, R8
```

Adição de inteiros (*Integer addition*)

addi reg1, reg2, reg3

Associa o valor inteiro, resultante da soma entre os registradores reg2 e reg3, ao registrador reg1.

Exemplo de uso:

```
addi R4, R2, R3
```

Adição em ponto flutuante (*floating point addition*)

addf reg1, reg2, reg3

Associa um valor com ponto flutuante, resultante da soma entre os registradores reg2 e reg3, ao registrador reg1.

Exemplo de uso:

```
addf R4, R2, R3
```

Operação OU bit-a-bit (*bitwise OR*)

or reg1, reg2, reg3

O registrador reg1 recebe os bits resultantes da operação OU (*OR*) aplicada, bit-a-bit, nos registradores reg2 e reg3.

Exemplo de uso:

```
OR R4, R2, R3
```

Operação E bit-a-bit (*bitwise AND*)

or reg1, reg2, reg3

O registrador reg1 recebe os bits resultantes da operação E (*AND*) aplicada, bit-a-bit, entre os registradores reg2 e reg3.

Exemplo de uso:

```
AND R4, R2, R3
```

Operação Ou exclusivo bit-a-bit (*bitwise exclusive or*)

XOR reg1, reg2, reg3

O registrador reg1 recebe os bits resultantes da operação OU-Exclusivo (*XOR*) aplicada, bit-a-bit, entre os registradores reg2 e reg3.

Exemplo de uso:

```
XOR R4, R2, R3
```

Rotação à direita (*rotate right*)

ror reg1, num

Realiza o deslocamento dos bits para a direita, sem perda, o número de vezes indicado por num.

Exemplo de uso:

```
ror R4, 3
```

Salta quando for igual (*Jump when equal*)

jmpEQ reg=R0, endereço

Salta para o endereço indicado quando o conteúdo do registrador reg for igual ao conteúdo do registrador R0. O endereço pode ser um número ou rótulo.

Exemplo de uso:

```
jmpEQ R4=R0, 42h
jmpEQ R2=R0, Fim
```

Salta quando for menor ou igual (*Jump when less or equal*)

jmpLE reg<=R0, endereço

Salta para o endereço indicado quando o conteúdo do registrador reg for menor ou igual ao conteúdo do registrador R0. O endereço pode ser um número ou rótulo.

Exemplo de uso:

```
jmpLE R4<=R0, 42h
jmpLE R2<=R0, Fim
```

Salto incondicional (*Unconditional jump*)

jmp endereço

Salta para o endereço indicado. O endereço pode ser um número ou rótulo.

Exemplo de uso:

```
jmp 42h
jmp Fim
```

Encerra a execução (*Stop program*)

halt

Exemplo de Código

```
          load    R1,Texto     ;endereço inicial do texto
          load    R2,1         ;valor a ser incrementado a cada passo
          load    R0,0         ;valor que indica o final da string
ProxCar:  load    RF,[R1]      ;obtém caractere e imprime na tela
          addi    R1,R1,R2     ;incrementa o endereço (próximo caractere)
          jmpEQ   RF=R0,Fim    ;quando o valor indicar o final do texto, então fim
          jmp     ProxCar      ;próximo caractere
Fim:      halt

Texto:    db      10           ; 10 indica quebra de linha (enter)
          db      "Hello world !!",10
          db      "    usando o ",10
          db      "  Simple Simulator",10
          db      0            ;final do texto
```

Exercícios

1) Desenvolva um programa que efetue a soma de um conjunto de valores inteiros positivos. Considere que o final da lista de valores seja indicado pelo valor 0, ou seja, para a lista 10,2,5,74,6,0 o último valor, que é zero deverá encerrar a soma dentro do programa.

2) Faça um programa que dados dois números seja efetuada a subtração entre eles. Lembre-se de como funciona a subtração em Complemento de 2! Além disso, para que seja possível inverter os bits são necessárias as operações lógicas bit-a-bit.

3) Faça um programa que ao receber dois valores inteiros efetue a multiplicação entre os mesmos. Lembre-se de que não existe no **SimpSIM** a instrução de multiplicação, então a mesma deve ser feita por meio de somas sucessivas!

4) Faça um programa que ao receber um conjunto de dados, por exemplo, a palavra “PRUDENTE” inverta os caracteres na memória.

Resolução:

4-

```
            load R0, Texto
            load R1, FimTexto
            load R3, 1      ;incremento (+1)
            load R4, -1     ;decremento (-1)
            addi R1, R1, R4 ; subtrai 1 do contador R1
Troca:      load R5,[R0]    ;guarda caractere indicado por R0 em R5
            load R6,[R1]
            store R6,[R0]
            store R5, [R1]
            addi R0, R0, R3
            addi R1, R1, R4
            jmpLE R1<=R0, Mostra
            jmp Troca
Mostra:     load R1, Texto
            load R0, 0
Loop:       load RF,[R1]
            addi R1,R1,R3
            jmpEQ RF=R0, Fim
            jmp Loop
Fim:        halt

Texto:      db "PRUDENTE"
FimTexto:   db 0
```