

Subrotinas

(Como implementar multiplicação e divisão uma vez só :-)

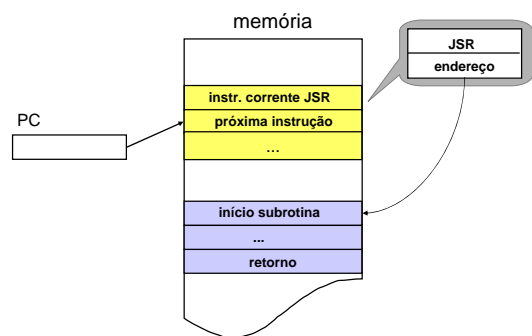
Subrotinas

- Características
 - função utilizada várias vezes
 - somente necessita ser codificada uma vez
 - vários pontos do programa de onde a rotina é “**chamada**”
 - vários pontos para onde a rotina deve “**retornar**”
 - rotina deve sempre retornar ao “ponto de chamada” correto

Chamada de uma rotina

- corresponde a um desvio do programa principal para o início da rotina
- imediatamente antes do desvio, o **pc** aponta para o “ponto de retorno” (instrução seguinte)
- este valor do **pc** deve ser armazenado, para possibilitar o retorno correto
- problema:
 - onde guardar o “ponto de retorno” ?

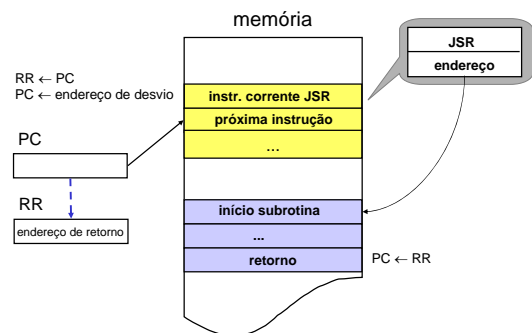
Chamada de rotina



Armazenamento do retorno: reg. especial

- em um registrador especial
 - por exemplo, RR, o “registrador de retorno”
 - desvio: $RR \leftarrow PC, PC \leftarrow \text{endereço}$
 - retorno: $PC \leftarrow RR$
- Vantagem: simplicidade
- Desvantagens:
 - sem aninhamento
 - uma rotina não pode chamar outra
 - sem recursividade
 - uma rotina não pode chamar a si mesma

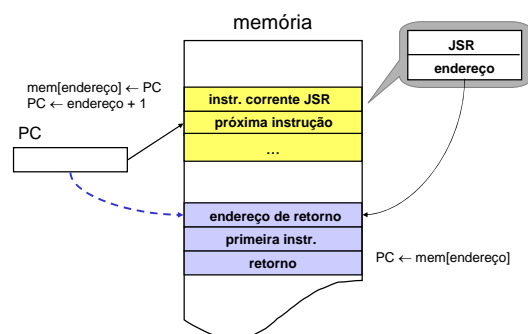
Registrador especial



Armazenamento do retorno: endereço especial

- em um endereço especial
 - por exemplo, o primeiro byte da sub-rotina
- desvio: $\text{mem}[\text{endereço}] \leftarrow \text{PC}$
 $\text{PC} \leftarrow \text{endereço} + 1$ corresponde ao endereço de desvio
- retorno: $\text{PC} \leftarrow \text{mem}[\text{endereço}]$; (o mesmo do desvio !)
- Vantagem: permite aninhamento
- Desvantagem: sem recursividade

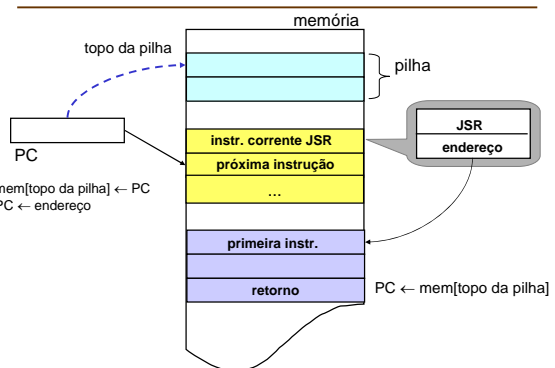
Endereço especial



Armazenamento do retorno: estrutura especial

- em uma estrutura especial
 - por exemplo, uma pilha
- desvio: $\text{mem}[\text{topo da pilha}] \leftarrow \text{PC}$
 $\text{PC} \leftarrow \text{endereço}$
- retorno: $\text{PC} \leftarrow \text{mem}[\text{topo da pilha}]$ (o mesmo do desvio !)
- Vantagens:
 - com aninhamento
 - com recursividade
- Desvantagem: gerência da pilha

Pilha

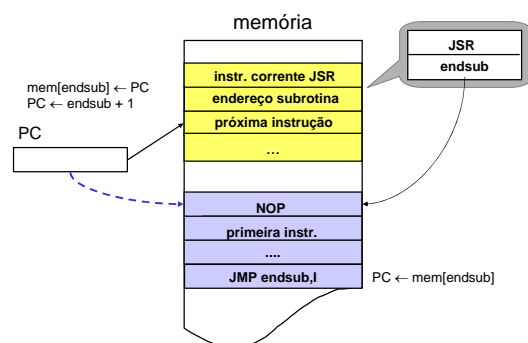


Ramses - subrotinas

- utiliza o primeiro byte da rotina
 - primeiro byte deve ser reservado ! (NOP)
- instrução JSR guarda PC de retorno neste byte
- primeira instrução começa no byte seguinte
- retorno feito de forma especial (JMP endsub,l)
- exemplo: se rotina está no endereço 100

desvio: JSR 100
 retorno: JMP 100,l

JSR no Ramses



Passagem de parâmetros

- Por registradores
- Por posições de memória
- exemplo: multiplicação
 - programa principal usa três posições:
 - primeiro_operando
 - segundo_operando
 - resultado
 - subrotina usa outras três posições:
 - op1
 - op2
 - resul

Passagem de parâmetros

- Por registradores

Programa principal

```
LDR A primeiro_operando
LDR B segundo_operando
JSR multiplica
STR A resultado
```

Subrotina

```
NOP
STR A op1
STR B op2
<multiplicação>
LDR A resul
JMP multiplica,l
```

Passagem de parâmetros

- Por posições de memória

Programa principal

```
LDR A primeiro_operando
STR A param1
LDR A segundo_operando
STR A param2
JSR multiplica
LDR A param3
STR A resultado
```

Subrotina

```
NOP
LDR A param1
STR A op1
LDR A param2
STR A op2
<multiplicação>
LDR A resul
STR A param3
JMP multiplica,l
```

Passagem de parâmetros

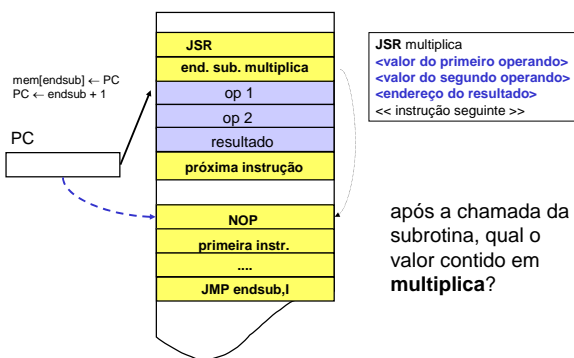
- Por posições de memória

Programa principal

```
JSR multiplica
<valor do primeiro operando>
<valor do segundo operando>
<endereço do resultado>
<< instrução seguinte >>
```



Manipulação de parâmetros



Passagem por posições de memória

```
JSR multiplica
<valor do primeiro operando>
<valor do segundo operando>
<endereço do resultado>
<< instrução seguinte >>
```

- manipulação dos parâmetros
 - usando endereçamento indireto (1)
 - usando indexado (2)
 - usando indexado com otimizações (3)

Manipulação de parâmetros (1)

usando modo indireto

multiplica: NOP
 LDR A multiplica,l ; obtém valor do primeiro operando
 STR A op1
 LDR A multiplica ; atualiza endereço de retorno
 ADD A #1
 STR A multiplica
 LDR A multiplica,l ; obtém valor do segundo operando
 STR A op2
 LDR A multiplica ; atualiza endereço de retorno
 ADD A #1
 STR A multiplica
 <multiplicação>
 LDR A multiplica,l ; obtém endereço do terceiro operando
 STR A ponteiro
 LDR B resul ; obtém resultado da rotina
 STR B ponteiro, l ; salva resultado no endereço do terceiro parâmetro.
 LDR A multiplica ; atualiza endereço de retorno
 ADD A #1
 STR A multiplica
 JMP multiplica, l ; retorna da subrotina

Manipulação de parâmetros (2)

usando modo indexado

multiplica: NOP
 LDR X multiplica ; X aponta para área de parâmetros
 LDR A 0,X ; obtém valor do primeiro operando
 STR A op1
 LDR A 1,X ; obtém valor do segundo operando
 STR A op2
 <multiplicação>
 LDR A 2,X ; obtém endereço do terceiro parâmetro
 STR A ponteiro ; ponteiro para guardar resultado
 LDR B resul ; obtém produto calculado pela subrotina
 STR B ponteiro, l ; salva resultado no endereço apontado
 ADD X #3
 STR X multiplica ; atualiza endereço de retorno
 JMP multiplica, l ; retorna da subrotina

Manipulação de parâmetros (3)

(otimizando ainda mais)

multiplica: NOP
 LDR X multiplica ; X aponta para área de parâmetros
 LDR A 0,X ; obtém valor do primeiro operando
 STR A op1
 LDR A 1,X ; obtém valor do segundo operando
 STR A op2
 <multiplicação>
 ADD X #3
 STR X multiplica ; atualiza endereço de retorno
 LDR X 255,X ; obtém endereço do terceiro parâmetro em MEM(RX - 1)
 LDR B resul ; obtém produto calculado pela subrotina
 STR B 0,X ; salva resultado no endereço do terceiro parâmetro
 JMP multiplica, l ; retorna da subrotina

Passagem de parâmetros

- por registradores
 - uso limitado pela quantidade de registradores disponíveis
- por posições de memória
 - localização dos parâmetros na memória depende de convenção estabelecida entre programa principal e subrotina
- por pilha
 - (aguardem o CESAR)