Guião de Laboratório
de
Arquitectura de Computadores

Simulação 5.7 – Ordenação por bolha com guarda de registos

1 – Objectivos

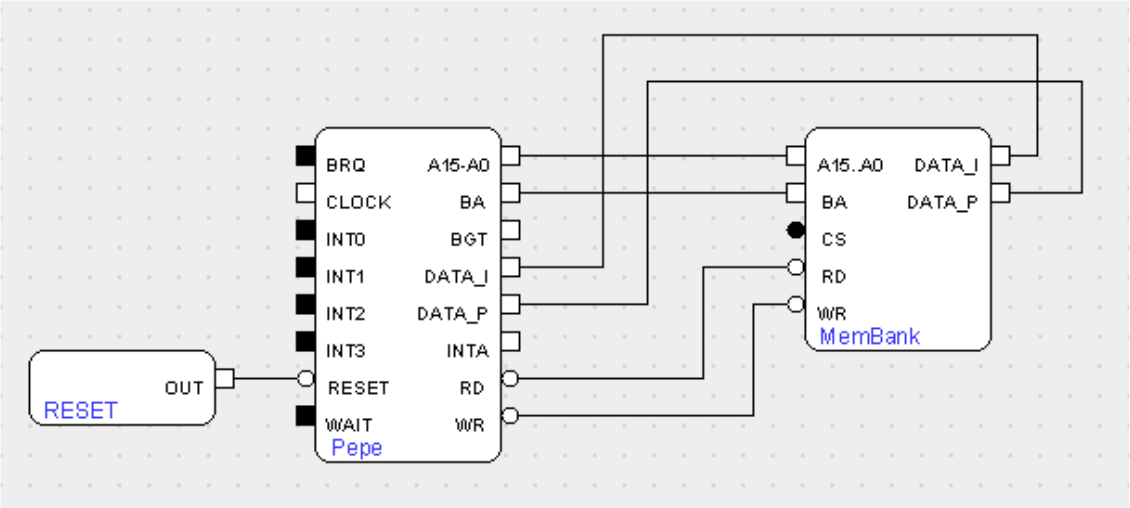
Esta simulação aplica o mecanismo de guarda/recuperação de registos nas rotinas ao programa de ordenação de números pelo método de bolha. O programa de base é o da Tabela 5.8, com as adaptações necessárias para incluir preservação de registos. A Tabela 5.23 mostra a rotina ordena da Tabela 5.8 já modificada para incluir este aspecto. O programa completo está incluído nesta simulação.

Os aspectos cobertos incluem os seguintes:

- Execução com pontos de paragem;
- Verificação do funcionamento das instruções PUSH e POP;
- Verificação da evolução dos registos relevantes e da pilha (em termos de conteúdo e de profundidade).

2 – Circuito

O ficheiro "pepe.cmod" implementa o circuito da Fig. 4.7. A simulação 4.1 contém indicações mais detalhadas sobre a sua utilização no simulador.



3 – Simulação do programa da tabela 5.23

Carregue este circuito no simulador e passe para Simulação.

Abra o painel do PEPE e compile e carregue ( ) o ficheiro "programa-tabela-5-23.asm".

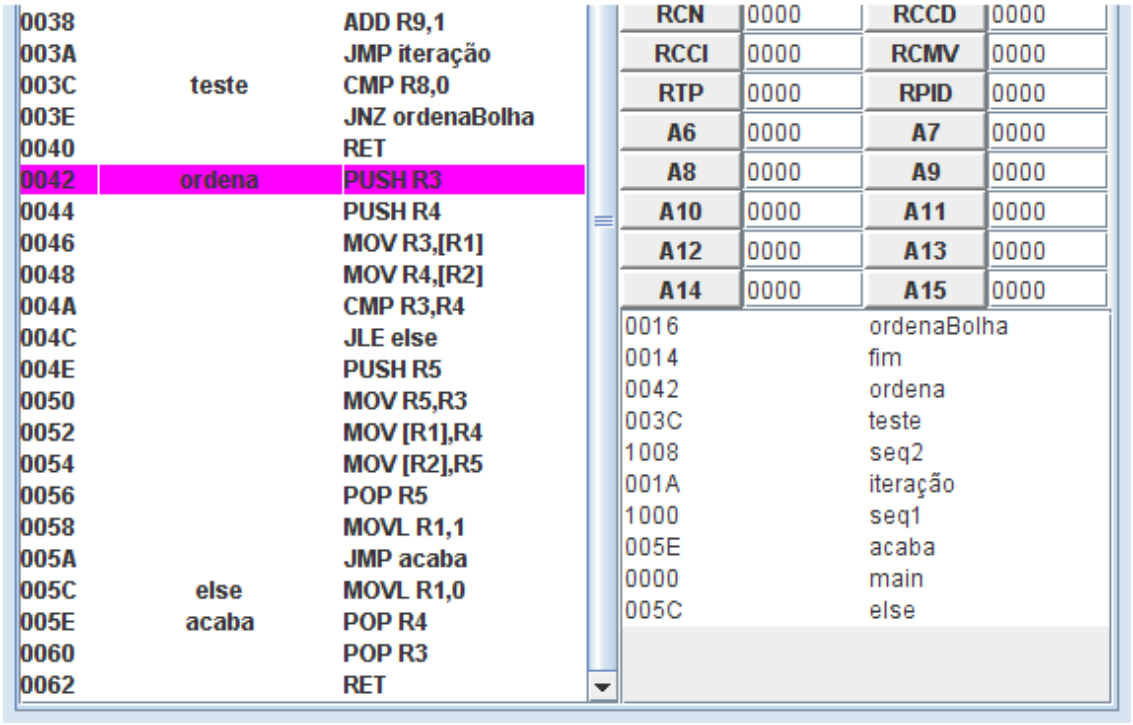
Este programa contém o programa de assembly da Tabela 5.8, alterado para incluir guarda de registos na rotina ordena. Todos os registos de utilização local a esta rotina R3, R4 e R5 são guardados e recuperados.

Note-se que a guarda/recuperação de R5 é feita apenas dentro da instrução if. Podia estar junta com a dos registos R3 e R4, mas assim evita-se perder tempo com o R5 caso a expressão booleana do if seja falsa. Verifica-se assim que a guarda/recuperação deve ser feita no âmbito mais restrito possível e não necessariamente a nível global da rotina.

Note-se também que esta rotina altera o R1 e este não é guardado/recuperado. É de propósito, pois corresponde ao valor retornado pela rotina.

NOTA – Na Tabela 5.23, falta a instrução MOV R1, 1 antes da instrução JMP acaba. Esta gralha está referenciada na errata do livro.

Coloque um ponto de paragem na primeira instrução da rotina ordena e execute em modo contínuo.



Quando parar, anote o valor dos registos R3, R4 e R5, e execute passo a passo até ao fim da rotina. Verifique que o valor inicial dos três registos é reposto, e que a ordem dos POPs é inversa à dos PUSHes.

4 – Variante ao programa da tabela 5.23

Na realidade, não era preciso guardar os registos R3, R4 e R5 porque no programa eles não são usados para mais nada. Tal, aliás, como os registos R1, R2, R8, R9 e R10 na rotina ordenaBolha.

No entanto, nem sempre os registos disponíveis chegam, e futuras alterações podem não reparar em todos os registos usados e estragar o valor de alguns desses registos. Como boa prática, qualquer rotina deve guardar e repor todos os registos cujo valor alterar.

Note-se que uma rotina não deve alterar (e por conseguinte não guardar/repor) os seus parâmetros (R6 e R7 em ordenaBolha e R1 e R2 em ordena).

Assim, o ficheiro “programa-tabela-5-23-variante.asm” faz as seguintes alterações face ao “programa-tabela-5-23.asm”:

- guarda também os registos usados pela rotina ordenaBolha (R1, R2, R8, R9 e R10)
- para mostrar a efectividade da guarda e recuperação dos registos, passa a (re)usar R8, R9 e R10 em vez de R3, R4 e R5, respectivamente, em ordena. Se não fosse a guarda e reposição dos registos, naturalmente o programa passaria a funcionar mal, pois ordena destruiria o valor dos registos de ordenaBolha.

Faça reset no simulador ou no PEPE. Compile e carregue () o ficheiro “programa-tabela-5-23-variante.asm”. Coloque um endereço de paragem na instrução “JMP fim” (senão o painel da memória não é actualizado).

Abra o painel da memória na zona do endereço 1000H e execute o programa em modo contínuo. Verifique que os arrays são ordenados, o que significa que o programa está a funcionar bem, e as rotinas estão assim melhor organizadas.

Note a diferença introduzida na rotina ordenaBolha:

```
; houveTroca usa o R8
; i usa o R9
; R10 é usado para valores temporários
      PUSH  R8          ; guarda o valor de R8
      PUSH  R9          ; guarda o valor de R9
      PUSH  R10         ; guarda o valor de R10
ronda:      MOV   R8, 0       ; houveTroca = false;
      MOV   R9, 0       ; i = 0;
iteração:
      MOV   R10, R7     ; cópia de n
      SUB   R10, 1      ; n-1
      CMP   R9, R10     ; i < n-1 ?
      JGE   teste       ; se não, o ciclo acabou
      MOV   R10, R9     ; obtém cópia de i
      SHL   R10, 1      ; 2*i (ender. em bytes)
      ADD   R10, R6     ; endereço de a [i]
      MOV   R1, R10     ; 1º parâmetro (&a[i])
      MOV   R10, R9     ; obtém cópia de i
      ADD   R10, 1      ; i+1
      SHL   R10, 1      ; 2*(i+1)
      ADD   R10, R6     ; endereço de a [i+1]
      MOV   R2, R10     ; 2º parâmetro (&a[i+1])
      CALL  ordena      ; chama a função
      OR    R8, R1      ; houveTroca || ordena
      ADD   R9, 1       ; i = i + 1;
      JMP   iteração    ; próxima iteração
teste:      CMP   R8, 0 ; houveTroca = falso ?
      JNZ   ronda       ; mais uma ronda
      POP   R10         ; repõe o valor de R10
      POP   R9          ; repõe o valor de R9
      POP   R8          ; repõe o valor de R8
      RET               ; retorna da função
```

Antes, o último JNZ saltava directamente para ordenaBolha. Devido aos PUSHes, que só devem ser executados uma vez no início da rotina, introduziu-se mais uma etiqueta (“ronda:”) para a qual agora o JNZ pode saltar.