Montador absoluto - Sistemas de Programação

Lucas Mafra Juliano Barros - 8993347
Rafael Camargos Santos - 8041564

PCS3616 - Prof. Ricardo Rocha

1 Introdução

Os montadores são programas de extrema relevância para a computação por, de forma resumida, permitir que programadores façam programas que executam tarefas mais complexas escrevendo menos linhas de código, através de abstrações (como mnemônicos) que são traduzidas em instruções baixo-nível para o processador.

Foi proposto nesta disciplina que se extendesse um montador elaborado por alunos no ano anterior de forma a comportar duas novas funcionalidades:

1) Representação do operando OI em base binaria, octa e ASCII (além de HEX)

2) Rótulos de tamanho 4 (duas palavras)

O presente relatório visa documentar a solução implementada pela dupla para atender aos novos requisitos propostos.

2 Observações sobre o código de 2018

É necessário apontar que, ao longo do processo de estudo do código fornecido e desenvolvimento da solução, notou-se que havia vários problemas com o projeto fornecido, dentre eles:

1) O operando OI era sempre tratado como valor numérico, mesmo quando consistia em um símbolo a ser resolvido a partir da tabela de símbolos.

2) A rotina ATTP não funcionava, zerando sempre o ponteiro PARG ao invés de atualizar os ponteiros desejados (PARG1, PARG2 E PARG3). Como consequência, a leitura do código ASM não era feita corretamente

3) O operando OI era sempre tratado como uma palavra apenas.

Esses foram apenas alguns dos problemas que se identificou durante o desenvolvimento. Infelizmente, não houve tempo de corrigir todos os problemas de forma que o montador usado como base está incorreto, não sendo assim possível testar a solução desenvolvida.

3 Extendendo o código para lidar com operando em outras bases

Para lidar com operandos em outras bases, adotou-se primeiro a seguinte definição de caracteres para representação destas bases:

Nome da base	Valor da base	Símbolo
ASCII	256	’
Hexadecimal	16	/
Decimal	10	=
Octal	8	@
Binário	2	#

Figure 1: Tabela com bases suportadas pelo montador em Java

Tendo definida a representação, criou-se constantes no arquivo "ctes.asm" para representar os caracteres definidos, conforme figura abaixo:

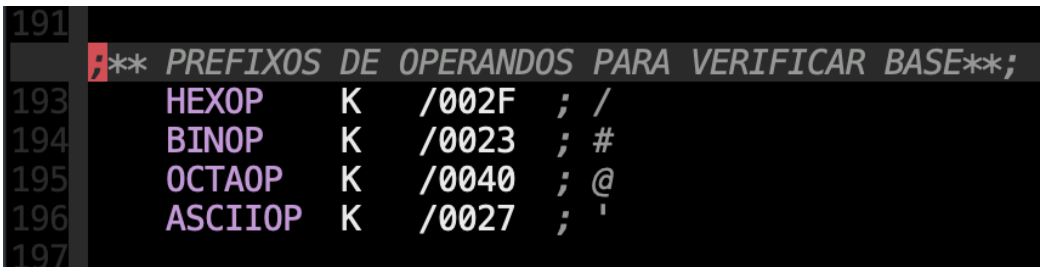

Figure 2: Constantes para representar as diferentes bases numéricas

Na sequência, alterou-se a rotina MONTA2 para identificar a base do operando e converter para hex antes de escrever a linha no arquivo de saída.

Os trechos mais importantes do código referente a alteração podem ser conferidos nas figuras abaixo:

```
MNTARG      $   =0001         ; CARREGA O OPERANDO
            MM  OPERAND
            SC  GET-PREFIX    ; PEGA O PREFIX DO OPERANDO PARA DESCOBRIR A BASE
            MM  OP-PREFIX
            -   HEXOP         ; VERIFICA SE OPERANDO ESTÁ EM HEX
            JZ  IN-HEXOP
            LD  OP-PREFIX
            -   BINOP
            JZ  IN-BINOP      ; VERIFICA SE OPERANDO ESTÁ EM BINARIO
            LD  OP-PREFIX
            -   OCTAOP
            JZ  IN-OCTAOP     ; VERIFICA SE OPERANDO ESTÁ EM OCTA
            LD  OP-PREFIX
            -   ASCIIOP
            JZ  IN-ASCIIOP    ; VERIFICA SE OPERANDO ESTÁ EM ASCII
```

Figure 3: Alterações na rotina MONTA2 para lidar com outros tipos de base para operando OI (1)

```
GET-PREFIX  $  =1
            SC UNPACK
            LD B1
            RS GET-PREFIX
IN-HEXOP    LD OPERAND          ; JA ESTÁ EM HEX, NADA A FAZER
            JP ESCREVE
IN-BINOP    LD OPERAND
            SC BINHEX              ; CONVERTE DE BINARIO PARA HEX (REPRESENTACAO ASCII), RESULTADO NO ACUMULADOR
            JP ESCREVE
IN-OCTAOP   LD OPERAND
            SC OB               ; CONVERTE DE OCTA PARA BINARIO, RESULTADO NO ACUMULADOR
            SC BINHEX           ; CONVERTE DE BINARIO PARA HEX (REPRESENTACAO ASCII), RESULTADO NO ACUMULADOR
            JP ESCREVE
IN-ASCIIOP  LD OPERAND
            SC CONV               ; CONVERTE DE ASCII PARA HEX, RESULTADO NO ACUMULADOR
            JP ESCREVE
ESCREVE     +   OPCODE        ; SOMA COM O OPERADOR
            PD  /0302         ; ADICIONA INSTRUÇÃO AO ARQUIVO MVN
            LD  NEWLINE       ; ADICIONA UMA NOVA LINHA
            PD  /0302         ; AO ARQUIVO
            RS  MONTALINHA
```

Figure 4: Alterações na rotina MONTA2 para lidar com outros tipos de base para operando OI (2)

A implementação das rotinas de conversão utilizadas no trecho mostrado pode ser conferida abaixo:

```
298 ; ====================================================================
299 ; ====================================================================
300 ;  *** BINHEX - ROTINA DE CONVERSÃO BINÁRIO PARA HEXADECIMAL ***  ©JJN/2005
301 ;               ENTRADA NO ACUMULADOR - NUMERO BINÁRIO DE 16 BITS
302 ;               RESULTADO: 4 CARACTERES ASCII NO ARQUIVO DE SAIDA
303 ;                                                                  OK CODIFICADO
    FATOR     K     /0000 ;
305
306     BINHEX  JP    /00         ; ENTRADA DE BINHEX
307
308             MM    AUX         ; SALVA NUMERO ORIGINAL
309             LD    H1000       ; FAZ FATOR=2**12
310
311     LPBHEX  MM    FATOR       ; POSICIONA NO PRÓXIMO NIBBLE A TRATAR
312
313             LD    AUX         ; RECUPERA O QUE RESTA DO NUMERO ORIGINAL
314             /     FATOR       ; OBTEM PROXIMO NIBBLE MAIS SIGNIFICATIVO
315             MM    AUX1        ; SALVA-O
316
317             SC    BH          ; CONVERTE PARA HEXADECIMAL (ASCII)
318             PD    /3          ; GERA-O NO ARQUIVO DE SAIDA
319
320             LD    AUX1        ; RECUPERA O ÚLTIMO NIBBLE TRATADO
321             *     FATOR       ; POSICIONA-O ADEQUADAMENTE
322             MM    AUX1        ; GUARDA PARA SUBTRAIR ADIANTE
323
324             LD    AUX         ; RECUPERA O RESTANTE DO NUMERO
325             -     AUX1        ; REMOVE O NIBBLE TRATADO
326             MM    AUX         ; GUARDA PARA TRATAR O PRÓXIMO NIBBLE
327
328             LD    FATOR       ; FAZ FATOR
329             /     H10         ; := FATOR/(2**4)
330             JZ    FIMBHEX     ; SE ESGOTOU OS 4 DIGITOS, VAI RETORNAR
331             JP    LPBHEX      ; SE NÃO, VOLTA AO LOOP
332
333     FIMBHEX RS    BINHEX      ; RETORNA
```

Figure 5: Implementação da rotina BINHEX (binario->hex)

4 Conclusão

Deve-se dizer que houve extrema dificuldade em realizar as implementações acima. Não é tão fácil adicionar-se novas funcionalidades a um código assembly feito por outra pessoa, mesmo com os comentários que auxiliam na compreensão do código. Olhando em retrospectiva, conclui-se que gastou-se talvez demasiado tempo tentando compreender o código e o que não estava funcionando ao invés de conformar-se com o montador da maneira em que estava e focar apenas em implementar as funcionalidades pedidas, de forma que não houve tempo hábil para implementar a segunda extensão (rótulo com tamanho 4). Ainda assim, conclui-se que foi um bom exercício para familiarizar-se com um projeto assembly e exercitar os conceitos aprendidos durante a disciplina.

```
; ===================================================================
; ROTINAS ADICIONAIS, A SEREM CODIFICADAS EM LINGUAGEM DE MÁQUINA
; ===================================================================
;
;** OB - CONVERTE UM DIGITO OCTAL PARA BINÁRIO - ©JJN/2005
;          RECEBE O DADO NO ACUMULADOR
;          RETORNA O RESULTADO NO ACUMULADOR E RESUL (SE INVALIDO, RETORNA -1)
;
; SEQUENCIA DE CHAMADA:
;                        LD INFASC  ; SUPONDO QUE INFASC TENHA UM DIGITO DECIMAL
;                        SC OB      ; OB CONVERTE-O PARA BINÁRIO, E
;                        MM INFBIN  ; O RESULTADO PODE SER GUARDADO EM INFBIN
                                   ; MAS TAMBÉM ESTÁ DISPONÍVEL EM RESUL

      OB    JP /00     ; ENTRADA DE OB
            MM DADO    ; SALVA EM DADO A INFORMAÇÃO RECEBIDA NO ACUMULADOR
            LD PPDOCT  ; ENDEREÇO DE DDEC (TABELA ASCII DE DIG. OCTAIS)
            MM PTX
            LD PDOCT   ; COMPRIMENTO DA TABELA DDEC
            MM CTX
            LD PPBOCT  ; ENDEREÇO DE BDEC (TABELA BINÁRIA DE DÍG. DECIMAIS)
            MM PTY
            SC TRADUZ1 ; CHAMA A SUBROTINA TRADUZ1
            RS OB      ; RETORNA COM O RESULTADO NO ACUMULADOR E RESUL
```

Figure 6: Implementação da rotina OB (octa->binario)

```
; ===================================================================
; ROTINAS ADICIONAIS, A SEREM CODIFICADAS EM LINGUAGEM DE MÁQUINA
; ===================================================================
;
;** OB - CONVERTE UM DIGITO OCTAL PARA BINÁRIO - ©JJN/2005
;          RECEBE O DADO NO ACUMULADOR
;          RETORNA O RESULTADO NO ACUMULADOR E RESUL (SE INVALIDO, RETORNA -1)
;
; SEQUENCIA DE CHAMADA:
;                        LD INFASC  ; SUPONDO QUE INFASC TENHA UM DIGITO DECIMAL
;                        SC OB      ; OB CONVERTE-O PARA BINÁRIO, E
;                        MM INFBIN  ; O RESULTADO PODE SER GUARDADO EM INFBIN
                                   ; MAS TAMBÉM ESTÁ DISPONÍVEL EM RESUL

      OB    JP /00     ; ENTRADA DE OB
            MM DADO    ; SALVA EM DADO A INFORMAÇÃO RECEBIDA NO ACUMULADOR
            LD PPDOCT  ; ENDEREÇO DE DDEC (TABELA ASCII DE DIG. OCTAIS)
            MM PTX
            LD PDOCT   ; COMPRIMENTO DA TABELA DDEC
            MM CTX
            LD PPBOCT  ; ENDEREÇO DE BDEC (TABELA BINÁRIA DE DÍG. DECIMAIS)
            MM PTY
            SC TRADUZ1 ; CHAMA A SUBROTINA TRADUZ1
            RS OB      ; RETORNA COM O RESULTADO NO ACUMULADOR E RESUL
```

Figure 7: Implementação da rotina CONV (ascii->hex)