Laboratório de Sistemas Processadores e Periféricos
Lista de comandos de Assembly

Gustavo G. Parma

Lista dos comandos assembly que serão utilizados ao longo das práticas.

1 Comandos

1. **ADD destino, fonte**

   Executa a adição direta entre dois operandos, o fonte e o destino. O resultado será armazenado no operando destino. Esta operação afeta os flags AF, CF, OF, PF, SF, ZF

2. **AND destino, fonte**

   Executa a operação lógica AND entre o destino e a fonte.O resultado fica armazenado no destino. Os falgs CF, OF, PF, SF e ZF são afetados por esta instrução.

3. **CALL alvo**

   Esta instrução é utilizada para chamar uma sub-rotina, identificada pelo nome **alvo**. Após a execução da subrotina, o programa volta ao fluxo normal de execução do programa principal. Nenhum flag é afetado por essa instrução.

4. **CLD**

   Limpa o bit DF no registrador de flags. Após esta instrução o valor de DF será igual a zero. Desta forma, ao se utilizar uma instrução de string, os registradores de índice serão incrementados automaticamente.

5. **CMP destino, fonte**

   Esta instrução realiza a subtração entre os dois operandos. Entretanto ela não retorna o valor do resultado, alterando apenas os flags de sinalização. Pode-se, portanto, saber qual a relação entre os valores comparados pela verificação de *ZF* e *CF*.

ZF=1	Os dois operandos são iguais
CF=1	O destino é menor do que a fonte
CF=0	O destino é maior ou igual Ã fonte

   Os seguintes flags são afetados por esta instrução: AF, CF, OF, PF, SF, ZF.

   Após esta instrução, pode-se utilizar qualquer um dos saltos condicionais (jump), observando-se se os números comparados são sinalizados ou não.

6. **CMPS**

   CMPSB

   CMPSW

   Esta instrução faz a comparação entre strings, efetuando a subtração entre o byte (ou palavra) endereçado por DI (dentro do segmento extra - ES) e o byte (ou palavra) endereçado por SI (dentro do segmento de dados - DS). A instrução automaticamente incrementa ou decrementa os registradores DI ou SI, dependendo do valor do flag DF. O valor de decremento ou incremento depende da instrução utilizada (CMPSB incrementa/decrementa de um unidade (byte) e CMPSW incrementa/decrementa de duas unidades (word)).

   Pode-se utilizar esta instrução em conjunto com as instruções REPE ou REPNE. Após esta instrução pode-se utilizar as instruções de salto condicional (jump).

7. **DEC destino**

   Esta instrução subtrai de uma unidade o operando, afetando os flags AF, OF, PF, SF e ZF.

8. **DIV fonte**

   Realiza a divisão entre o registrador AX(numerador) e a fonte(denominador), se a fonte for de 8 bits. Neste caso o quociente será armazenado em AL e o resto em AH.

   Se a fonte for de 16 bits, o numerador(32 bits) deve estar armazenado no conjunto DX-AX, sendo que o quociente será armazenado em AX e o resto da divisão em DX.

   Esta operação afeta o flags CF e OF. O operando fonte tem que ser um registrador ou uma posição de memória.

9. **IN AX, endereço**

   IN AX, DX

   Em todos os casos, pode-se utilizar AL no lugar de AX. Utilizando-se AL, apenas um byte será lido no endereço fornecido. No primeiro caso (IN AX,endereço), o endereço poderá variar de $00h$ Ã FFh. Caso se utilize DX, os endereços poderão variar, consequentemente, de $0000h$ Ã $FFFFh$. Caso seja utilizado o registrador AX, o valor lido no endereço fornecido será carregado em AL e o valor lido no endereço subsequente será carregado em AH.

10. **INC destino**

    Adiciona 1 ao conteúdo de um operando, podendo ser um registrador ou posição de memória. Os flags afetados são: AF, OF, PF, SF e ZF.

    Para se incrementar o conteúdo de uma memória, deve-se utilizar: INC byt ptr[SI]. Neste caso, à posição de memória apontada por SI será adicionado uma unidade.

11. **INT tipo**

    Esta instrução altera o fluxo do programa, chamando uma rotina de interrupção que é determinada pelo **tipo**. Os flags IF e TF são alterados, sendo colocados em nÃvel baixo (zero).

12. **INTO**

    Se o flag OF estiver igual a um, essa instrução gera uma interrupção tipo 4. Os flags IF e TF são colocados em nÃvel baixo.

13. **IRET**

    Esta instrução finaliza uma rotina de interrupção, recuperando da pilha o conteúdo dos registradores que foram automaticamente salvos (IP, CS e flags), retornando o processador ao ponto de parada antes do pedido de interrupção. Todos os flags são afetados.

14. **JMP alvo**

    Esta instrução provoca um salto incondicional no fluxo de processamento. De forma similar Ã s instrução de salto condicional, é necessário que a distância entre o salto e o alvo seja um valor entre -128 e $+127$ bytes.

15. **JCONDICAO label_alvo_curto**

    Provoca um desvio no processamento para o label se uma condição testada for encontrada. Utiliza-se, normalmente, os flags para o teste da condição. O label_alvo tem que estar na faixa de -128 a +127 bytes da próxima instrução devido a utilização de apenas um byte (considerando número com sinal) a ser adicionado ao IP para gerar o endereço alvo. A Tabela a seguir mostra as condições existentes para o salto condicional. Neta tabela, ABOVE e BELOW referem-se a números não sinalizados e GREATER e LESS a números sinalizados.

Instrução	Desvio relativo se	Condição testada
JA	above	CF=0 e ZF=0
JAE	above ou equal	CF=0
JB	below	CF=1
JBE	below ou equal	CF=1 ou ZF=1
JC	carry	CF=1
JE ou JZ	equal (zero)	ZF=1
JG	greater	ZF=0 e SF=OF
JGE	greater ou equal	SF=OF
JL	less	(SF xor OF)=1
JLE	less ou equal	(SF XOR OF)=1 OU ZF=1
JNA	not above	CF=1 ou ZF=1
JNAE	not above nor equal	CF=1
JNB	not below	CF=0
JNBE	not below nor equal	CF=0 e ZF=0
JNC	not carry	CF=0
JNE ou JNZ	not equal (not zero)	ZF=0
JNG	not greater	((SF xor OF) or ZF)=1
JNGE	not greater nor equal	(SF xor OF)=1
JNL	not less	SF=OF
JNLE	not less nor equal	ZF=0 e SF=OF
JNO	not overflow	OF=0
JNP ou JPO	not parity (par. odd)	PF=0
JNS	not sign	SF=0
JO	overflow	OF=1
JP ou JPE	parity (parity even)	PF=1
JS	sign	SF=1

16. **LEA destino, fonte**

    Carrega no registrador destino o endereço efetivo de um operando na memória.

17. **LOOP label_alvo**

    neste caso, o registrador contador CX é decrementado a cada vez que o loop for executado, permanecendo o programa no loop até que CX seja igual a zero.

18. **MOV destino, fonte**

    Copia o conteúdo do operando fonte para o operando destino, sem destruir o conteúdo do operando fonte. O registrador de segmento de código (CS) pode ser utilizado como fonte de dados, mas não como destino. Os modos de acesso a dados foram explorados na prática anterior, sendo que existem sete modos de transferência de dados:

    (a) do acumulador para a memória; Neste caso, o acesso a memória pode ser realizado por um acesso imediato ou utilizando-se os registradores SI, ou DI, ou BX. (MOV [SI],AL ;MOV [BX+DI],AX; MOV [00A0h],DL

    (b) da memória para o acumulador;

    (c) da memória ou registrador para registrador de segmento

    (d) do registrador de segmento para a memória ou registrador

    (e) de registrador para registrador

    (f) dado imediato para registrador

    (g) dado imediato para memória

    quando for utilizar um dado imediato, o valor a ser transferido deve começar sempre com um número, ou seja, para se transferir FFh para o registrador AL, deve-se utilizar MOV AL, 0FFh.

19. **MUL fonte**

    Realiza a multiplicação entre a fonte e o registrador AX. Se a fonte for de 8 bits, a operação será realizada entre a fonte e o registrador AL. O resultado estará em AX. Se a fonte for de 16 bits, a operação será realizada entre a fonte e o registrador AX. A parte alta do resultado estará em DX e a parte baixa em AX.

    Esta operação afeta o flags CF e OF. O operando fonte tem que ser um registrador ou uma posição de memória.

20. **NOP**

    Esta operação não realiza nenhuma operação específica, sendo utilizada para propósitos de temporização.

21. **OR destino, fonte**

    Executa a operação lógica OR entre o destino e a fonte. O resultado fica armazenado no destino. Os falgs CF, OF, PF, SF e ZF são afetados por esta instrução.

22. **OUT endereço, AX**

    OUT DX, AX

    A instrução OUT opera de forma semelhante Ã instrução IN. Desta forma, pode-se optar por utilizar todo o registrador AX ou somente o registrador AL. Caso se utilize todo o registrador AX, o valor de AL será transferido para o endereço fornecido e o valor de AH será transferido para o endereço fornecido acrescido de uma unidade.

23. **POP destino**

    A instrução POP retira uma palava (16 bits) da pilha e a armazena no registrador destino.

24. **PUSH fonte**

    A instrução PUSH armazena o valor do registrador fonte (16 bits) na pilha.

25. **PROC** e **ENDP**

    Estas instruções não fazem parte do conjunto de instruções do assembly, mas são pseudo-operadores utilizados pelo compilador para delimitar uma sub-rotina. Ao se definir uma subrotina é necessário informar se essa subrotina é do tipo near ou far (dentro do próprio segmento de código ou entre segmentos, respectivamente).

    A seguinte listagem exemplifica a definição de uma subrotina:

    alvo PROC NEAR

        mov ax,0FFFAh

        mov bl,ah

        ret

    alvo ENDP

26. **REPE comando**

    REPNE comando

    Estes comandos são prefixos utilizados para repetir uma instrução de string pelo número de vezes especificado em CX. Após a execução do comando, o valor de CX é decrementado, sem afetar o estado dos flags de sinalização. REPE repete o comando enquanto $CX <> 0$ e $ZF = 1$. REPNE repete o comando enquanto $CX <> 0$ e $ZF = 0$.

27. **RET**

    Esta instrução encerra uma subrotina, fazendo com que o fluxo de processamento volte ao programa principal.

    A instrução CALL e RET funcionam de maneira parecida Ã s instruções INT e IRET. Entretanto, a subrotina existe no programa que está sendo executado e não em um local determinado

pelo vetor de interrupção. Portanto, é importante que, ao final da sub-rotina, o valor de SP esteja no mesmo nÃvel que estava na entrada da sub-rotina pois a função RET busca na pilha o valor de IP (ou IP e CS, no caso de subrotina do tipo inter-segmento) para que o programa possa voltar ao seu fluxo normal.

28. **ROL destino,1**

    ROL destino, CL

    Esta operação rotaciona o conteudo especificado pelo destino para a esquerda. O valor do bit mais significativo é colocado no CF e no lugar do bit menos significativo. A quantidade de bits a serem rotacionados é 1 (um) ou o número especificado em CL.

29. **ROR destino,1**

    ROR destino, CL

    Esta operação rotaciona o conteudo especificado pelo destino para a direita. O valor do bit menos significativo é colocado no CF e no lugar do bit mais significativo. A quantidade de bits a serem rotacionados é 1 (um) ou o número especificado em CL.

30. **SAL destino, 1**

    SAL destino, CL

    Esta operação desloca o conteúdo especificado pelo destino para a esquerda. O valor do bit mais significativo é colocado apenas no CF e é inserido um zero no lugar do bit menos significativo. A quantidade de bits a serem deslocados é 1 (um) ou o número especificado em CL.

31. **SAR destino,1**

    SAR destino,CL

    Esta operação desloca o conteúdo especificado pelo destino para a direita. O valor do bit menos significativo é colocado apenas no CF, sendo que o bit mais significativo permanece com o mesmo valor anterior. A quantidade de bits a serem deslocados é 1 (um) ou o número especificado em CL.

32. **STD**

    Seta o bit DF no registrador de flags. Após esta instrução, o valor de DF será igual Ã um. AO se realizar uma operação de string, os registradores de Ãndice são automaticamente decrementados.

33. **SUB destino, fonte**

    Executa a subtração direta entre dois operandos, o destino e o fonte (destino = destino - fonte). O resultado será armazenado no operando destino. Esta operação afeta os flags AF, CF, OF, PF, SF, ZF

34. **XCHG destino, fonte**

    Esta instrução faz a permutação entre o conteúdo dos dois operandos, que podem ser dois registradores ou um registrados e uma posição de memória. Os flags de sinalização não são alterados quando esta instrução é executada.

35. **XOR destino, fonte**

    Executa uma operação de ou-exclusivo entre o destino e a fonte. O resultado fica armazenado no destino. Os flags CF, OF, PF, SF e ZF são afetados por esta instrução.