

EEE 410 – Microprocessors I

Fall 05/06 – Lecture Notes # 7

Outline of the Lecture

- Control transfer instructions
- CALL statement
- Subroutines
- Data Types and Data Definition

CONTROL TRANSFER INSTRUCTIONS

FAR and NEAR : (given CS:IP)

- In NEAR the control is transferred within the current code segment (intra-segment). IP is changed.
- In FAR the control is transferred outside the current code segment (inter-segment). Both CS and IP are changed.

Conditional Jumps

In the conditional jump, control is transferred to a new location if a certain condition is met. JC (jump if carry) [CF is checked], JNZ (jump if not zero) [ZF is checked].

Mnemonic	Condition Tested	“Jump if ...”
JA/JNBE	(CF=0) and (ZF=0)	above/not below nor equal
JAE/JNB	CF=0	above or equal/not below
JB/JNAE	CF=1	below/not above nor equal
JBE/JNA	(CF or ZF)=1	below or equal/not above
JC	CF=1	carry
JE/JZ	ZF=1	equal/zero
JG/JNLE	((SF xor OF) or ZF) = 0	greater/not less nor equal
JGE/JNL	((SF xor OF) = 0	greater or equal/not less
JL/JNGE	((SF xor OF) = 1	less/not greater nor equal
JLE/JNG	((SF xor OF) or ZF) = 1	less or equal/not greater
JNC	CF=0	not carry
JNE/JNZ	ZF=0	not equal/not zero
JNO	OF=0	not overflow
JNP/JPO	PF=0	not parity/parity odd
JNS	SF=0	not sign
JO	OF=1	overflow
JP/JPE	PF=1	parity/parity equal
JS	SF=1	sign

Note: “above” and “below” refer to the relationship of two unsigned values; “greater” and “less” refer to the relationship of two signed values.

Short Jumps

- All conditional jumps are short jumps. In a short jump the address of the target must be within -128 to +127 bytes of the memory
- A conditional jump is a two-byte instruction: one byte is the opcode of the J condition and the second value is a value between 00 to FF.
- In a backward jump the second byte is the 2’s complement of the *displacement* value.

Ex:	:	:	:
	:	:	:
AGAIN:	ADD AL,[BX]	1067:000D 0207	ADD AL,[BX]
	INC BX	1067:000F 43	INC BX
	DEC CX	1067:0010 49	DEC CX
	JNZ AGAIN	1067:0011 75FA	JNZ 000D
	MOV SUM,AL	1067: <u>0013</u> A20500	MOV [0005],AL
	:	:	:

The instruction “JNZ AGAIN” is assembled as “JNZ 000D” and the 000D is the address of the instruction with label AGAIN.

(000D = 0013+FA = 000D) the carry is dropped. Note that FA is 2’s complement of –6, meaning that the address of target is –6 bytes from the IP of the next instruction.

Ex:	0005 8A 47 02	AGAIN:	MOV AL,[BX]+2
	0008 3C 61		CMP AL,61H
	000A 72 06		JB NEXT
	<u>000C</u> 3C 7A		CMP AL,7AH
	000E 77 02		JA NEXT
	<u>0010</u> 24 DF		AND AL,DFH
	0012 88 04	NEXT:	MOV [SI],AL

The displacement value is added to the IP of the next instruction to calculate the jump address.
 (06 + 000C = 0012) (02 + 0010 = 0012)

Unconditional Jumps

“JMP label” is the unconditional jump in which control is transferred unconditionally to the target label. Unconditional jump can take the following forms.

1. SHORT JUMP, which is specified by the format “JMP SHORT label” . In this jump the address of the target location is within –128 to +127 bytes of the current IP. It works like the conditional jump.
2. NEAR JUMP, which is the default has the format “JMP label”. It jumps within the current code segment. It is exactly same as the short jump, except that the target address can be anywhere within the range of +32767 to –32768.
3. FAR JUMP, has the format “JMP FAR PTR label”. This is the jump out of the current code segment, meaning that not only the IP, but also the CS is replaced with new values.

CALL STATEMENTS

- Another control transfer instruction is the CALL instruction, which is used to call a **procedure**
- The target address can be in the current segment, hence a NEAR call (IP is changed CS is not)
- The target address can be outside the current segment, hence FAR call (IP and CS are changed)
- To make sure that after the execution of the called subroutine the microprocessor knows where to come back, the microprocessor automatically saves the address of the instruction following the call on the stack.
- The last instruction of the called subroutine must be RET (return) instruction, which directs CPU to pop the address of the next instruction before the called subroutine to be restored

✚ SUBROUTINES

In Assembly Language there can be one main program and many subroutines called from the main program. Subroutines are organized as procedures. PROC can be FAR or NEAR. If not mentioned, by default a PROC is NEAR

Shell of the Assembly Language Subroutines.

```

;-----
CODSEG    SEGMENT
MAIN      PROC FAR
          ASSUME .....
          MOV  AX,...
          MOV  DA,AX
          CALL SUBR1
          CALL SUBR2
          CALL SUBR3
          MOV  AH,4CH
          INT  21H

MAIN      ENDP
;-----
SUBR1     PROC
          ...
          RET
SUBR1     ENDP
;-----
SUBR2     PROC
          ...
          RET
SUBR2     ENDP
;-----
SUBR3     PROC
          ...
          RET
SUBR3     ENDP
;-----
CODSEG    ENDS
          END          MAIN

```

Alternatively the following format can be used in **Simplified Segment Definition:**

```

;-----
.CODE
MAIN:     MOV  AX,@DATA
          MOV  DS, AX
          CALL SUBR1
          CALL SUBR2
          CALL SUBR3
          MOV  AH,4CH
          INT  21H

;-----
SUBR1:    ...
          RET

;-----
SUBR2:    ...
          RET

;-----
SUBR3:    ...
          RET
          END MAIN

```

✚ DATA TYPES AND DATA DEFINITION

➤ Assembler Data Directives

- **ORG (Origin)** : ORG is used to indicate the beginning of the offset address. The number after ORG can be either in hex or decimal. Mostly used in Data segment.

```

Ex:  DTSEG    SEGMENT
      DATA_IN DW  234DH,1DE6H,3BC7H,566AH
      SUM      DW  ?
      DTSEG    ENDS

```

- **DB (define byte)** : DB allows allocation of memory in bytes for decimal, binary, hex and ASCII.

```

Ex:  DATA1  DB  25          ;DECIMAL D IS OPTIONAL

```

```

DATA2    DB    10001001B    ;BINARY
DATA3    DB    12H          ;HEX
          ORG    0010H
DATA4    DB    '2591'       ;ASCII NUMBERS
          ORG    0018H
DATA5    DB    ?           ;SET ASIDE A BYTE
          ORG    0020H
DATA6    DB    'My name is Joe' ;ASCII CHARACTERS

```

- **DUP (duplicate)** : DUP is used to duplicate a given number of characters.

```

Ex:  DATA1    DB    0FFH,0FFH,0FFH,0FFH    ;FILL 4 BYTES WITH FF
      DATA2    DB    4 DUP(0FFH)          ;FILL 4 BYTES WITH FF

```

```

Ex:  DATA3    DB    32 DUP (?)             ;SET ASIDE 32 BYTES
      DATA4    DB    5 DUP (2 DUP (99))    ;FILL 10 BYTES WITH 99

```

- **DW(define word)** : DW is used to allocate 2 bytes (one word) of memory at a time. DUP is used to duplicate a given number of characters.

```

Ex:  DATA1    DW    954                    ;DECIMAL
      DATA2    DW    100101010100B        ;BINARY
      DATA3    DW    253FH                ;HEX
      DATA4    DW    9,2,7,0CH,00100000b,5,'HI' ;MISCELLANEOUS
      DATA5    DW    8 DUP (?)

```

- **EQU(equate)** : EQU is used to define a constant without occupying a memory location.

```

Ex:  COUNT    EQU    25
      MOV     CX,COUNT

```

- **DD(define doubleword)** : Same as DW but allocates 2 words (4 bytes) of memory at a time.

```

Ex:  DATA1    DD    1023                    ;DECIMAL
      DATA2    DD    100011100101010100B    ;BINARY
      DATA3    DD    5C2A57F2H              ;HEX
      DATA4    DD    23H,24789H,65533        ;MISCELLANEOUS

```

- **DQ(define quadword)** : DQ is used to allocate memory 8 bytes (4 words) in size.

```

Ex:  DATA1    DQ    4523C2H                ;HEX
      DATA2    DQ    'HELLO'                ASCII CHARACTERS
      DATA3    DQ    ?                       ;NOTHING

```

- **DT(define ten bytes)** : Allocates 10 bytes of memory space. Mostly used for BCD numbers.

```

Ex:  DATA1    DT    867943569829           ; Default is BCD not decimal

```