Advanced Microprocessors

Notes #4
Programming the Microprocessor - Part II

EE 467/567
Winter 2012
Avinash Kodi

Background Materials

• Textbook:
  - Sections 6.1, 6.2, 6.3
• Other:
  - "IA-32 Intel® Architecture Software Developer's Manual Volume 2a & 2b"
    • Where Applicable
Flag and Compare Instructions

Flags:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Flags/Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAHF</td>
<td>Load AH from flags</td>
<td>(AH) ← (Flags)</td>
</tr>
<tr>
<td>SAHF</td>
<td>Store AH into flags</td>
<td>(Flags) ← (AH), SF,ZF,AF,PF,CF</td>
</tr>
<tr>
<td>CLC</td>
<td>Clear carry flag</td>
<td>(CF) ← 0, CF</td>
</tr>
<tr>
<td>STC</td>
<td>Set carry flag</td>
<td>(CF) ← 1, CF</td>
</tr>
<tr>
<td>STI</td>
<td>Set interrupt flag</td>
<td>(IF) ← 1, IF</td>
</tr>
<tr>
<td>CLI</td>
<td>Clear interrupt flag</td>
<td>(IF) ← 0, IF</td>
</tr>
</tbody>
</table>

Compare:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Flags/Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMP</td>
<td>Compare</td>
<td>(D) - (S) is used in setting and resetting of the flags</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OF,SF,ZF,AF,PF,CF</td>
</tr>
</tbody>
</table>

Comparing Bits

MOV AX, 1226H
CMP AX, B234H
LAHF

Question: what is the contents of AX after execution of these instructions?

After instructions 1 & 2: (AX) = 0001 0010 0010 0110

<table>
<thead>
<tr>
<th>CF</th>
<th>SF</th>
<th>ZF</th>
<th>AF</th>
<th>PF</th>
<th>OF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

No borrow needed => AF = 0

Odd number of ones, thus
PF = 0

After instruction 3: (AX) = 0000 0011 0010 0110

Not affected
Transfer of Control

- **Jump or transfer of control** refers to a change in the execution path within a program,

- 

  - (CS) and (IP) determine which instruction is going to be *executed* next (CS:IP),
  - Transfer of control or jump, therefore, changes the contents of CS and IP.

---

Transfer of Control

*Decision Making & Branching - 1*

Branching / Jumping / Transfer of Control:
refers to a change in the execution path within a program
(jump to a different instruction of the program)

- Unconditional branching
  - Transfer of control or jumping within the program takes place no matter what.

- Conditional branching
  - Transfer of control or jumping takes place if and only if a specific condition is satisfied.
  - These conditions are often coupled to one or more flags in the status register.
  - \(c, z, n, v\)

The user program must set up one or more of these flags to be tested by the branch instruction!
Transfer of Control  
*Decision Making & Branching - 2*

**Transfer of Control**

**Jumping**

- **Intrasegment**
  - Address of the next instruction is located within the same segment. 
  - *Change only IP.*
  
  **Operands:**
  - Short-label: Signed-displacement -8-bit number
  - Address within CS - 16-bit number as immediate operand
  - Near-label: Signed-displacement -16-bit number
  - Address within CS - 16-bit number as immediate operand
  - Memptr16: Address within CS – Use any form of memory addressing
  - Regptr16: Address within CS – Use the contents of a register

- **Intersegment**
  - Address of the next instruction is located within another segment
  - *Change both CS and IP.*
  
  **Operands:**
  - Far-label: (IP) low word of 32-bit immediate operand
  - (CS) high word of 32-bit immediate operand
  - Memptr32: Location of the 32-bit (double word) is specified by any memory addressing mode.
  - Again (IP) – low word, (CS) – high word.

---

4.7

4.8
Transfer of Control

Examples - 1

Ex. 1: JMP CX (RegPtr16)
Before: (CS) = 0800, (IP) = 0040, (CX) = 0600
After: (CS) = 0800, (IP) = 0600, (CX) = 0600

Ex. 2: JMP 1002H
Before: (CS) = 0800, (IP) = 0040
After: (CS) = 0800, (IP) = 1002

The offset is encoded as either a signed 8-bit constant (short-label) or a signed 16-bit constant (Near-label), depending on the size of the difference.

In machine-code the difference equals 1002 – 0040 = 0FC2 -> 16-bits => Near-label

Ex. 3: JMP 0012ABCDH (Far-label)
Before: (CS) = 0800, (IP) = 0040
After: (CS) = 0012, (IP) = ABCD

Ex. 4: JMP DWORD PTR [SI] (MemPtr32)
Before: (CS) = 0800, (IP) = 0040, (DS)=0A00, (SI) = 0600, (0A600-3) = 03214567H
After: (CS) = 0321, (IP) = 4567

Ex 5: JMP NEXTINT
  
  NEXTINT: MOV etc. etc.

Depending on the location of this instruction determined after the first assembler pass a signed displacement is calculated during the second assembler pass (two-pass assembler).
Transfer of Control
Conditional Jump Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Condition</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>JA¹</td>
<td>Above</td>
<td>CF=0 and ZF=0</td>
</tr>
<tr>
<td>JAE¹</td>
<td>Above or equal</td>
<td>CF=0</td>
</tr>
<tr>
<td>JB¹</td>
<td>Below</td>
<td>CF=1</td>
</tr>
<tr>
<td>JBE¹</td>
<td>Below or equal</td>
<td>CF=1 or ZF=1</td>
</tr>
<tr>
<td>JC</td>
<td>Carry</td>
<td>CF=1</td>
</tr>
<tr>
<td>JCXZ</td>
<td>CX register is 0</td>
<td>(CF or ZF) = 0</td>
</tr>
<tr>
<td>JE¹²</td>
<td>Equal</td>
<td>ZF=1</td>
</tr>
<tr>
<td>JG¹</td>
<td>Greater</td>
<td>ZF=0 and SF=OF</td>
</tr>
<tr>
<td>JGE¹</td>
<td>Greater or equal</td>
<td>SF=OF</td>
</tr>
<tr>
<td>JL¹</td>
<td>Less</td>
<td>(SF xor OF)=1</td>
</tr>
<tr>
<td>JLE¹</td>
<td>Less or equal</td>
<td>((SF xor OF) or ZF)=1</td>
</tr>
<tr>
<td>JNA¹</td>
<td>Not above</td>
<td>CF=1 or ZF=1</td>
</tr>
<tr>
<td>JNAE¹</td>
<td>Not above or equal</td>
<td>CF=1</td>
</tr>
<tr>
<td>JNB¹</td>
<td>Not below</td>
<td>CF=0</td>
</tr>
<tr>
<td>JNBE¹</td>
<td>Not below or equal</td>
<td>CF=0 and ZF=0</td>
</tr>
<tr>
<td>JNC</td>
<td>Not carry</td>
<td>CF=0</td>
</tr>
<tr>
<td>JNE¹²</td>
<td>Not equal</td>
<td>ZF=0</td>
</tr>
<tr>
<td>JNG²</td>
<td>Not greater</td>
<td>((SF xor OF) or ZF)=1</td>
</tr>
<tr>
<td>JNLE¹</td>
<td>Not greater nor equal</td>
<td>(SF xor OF)=1</td>
</tr>
<tr>
<td>JNL²</td>
<td>Not less</td>
<td>SF=OF</td>
</tr>
<tr>
<td>JNO</td>
<td>Not overflow</td>
<td>OF=0</td>
</tr>
<tr>
<td>JNP</td>
<td>Not parity</td>
<td>PF=0</td>
</tr>
<tr>
<td>JNS</td>
<td>Not sign</td>
<td>SF=0</td>
</tr>
<tr>
<td>JNZ</td>
<td>Not zero</td>
<td>ZF=0</td>
</tr>
<tr>
<td>JO</td>
<td>Overflow</td>
<td>OF=1</td>
</tr>
<tr>
<td>JP</td>
<td>Parity</td>
<td>PF=1</td>
</tr>
<tr>
<td>JPE</td>
<td>Parity even</td>
<td>PF=1</td>
</tr>
<tr>
<td>JPO</td>
<td>Parity odd</td>
<td>PF=0</td>
</tr>
<tr>
<td>JS</td>
<td>Sign</td>
<td>SF=1</td>
</tr>
<tr>
<td>JZ</td>
<td>Zero</td>
<td>ZF=1</td>
</tr>
</tbody>
</table>

¹Refer to unsigned numbers (above and below)
²Refer to signed numbers (less and greater)

Note that all the status flags are set up by the instructions preceding the conditional jump.

Example: Compare the contents of registers SI and DI and fill AX with FFFF if (SI) > (DI), and with 0000 if (DI) >= (SI).

Note: the user assumes signed integers !!!

```
CMP   SI,DI
JG    FILL
MOV   AX,0000H
JMP   END
FILL: MOV   AX,FFFFH
END:   NOP
END
```
Transfer of Control

Conditional Jump Instructions

- **Example:** We compare two 64-bit integers at ML VAR1 and VAR2. Jump to a location called TEMP if the number at VAR1 is greater than or equal to the number at VAR2. (DS) = 0000H. Assume 32-bit registers are available.

MOV EAX,[VAR1]
MOV EBX,[VAR2]
SUB EAX,EBX
MOV EAX,[VAR1 + 4]
MOV EBX,[VAR2 + 4]
SBB EAX,EBX
JGE TEMP

Instructions do not affect flags

**NASM allows you to do this!!**

Transfer of Control

Structures - 1

Branch:

IF .... THEN .... ELSE

Example:

AND AL,08H ← Setup condition
IZ BIT_HIGH ← Test condition

............

............

BIT_HIGH: .......... ← Label indicates where to jump to
Transfer of Control
Structures - 2

Loop:
REPEAT .... UNTIL

- Initialize counter or Initialize condition
- Loop statements
- Decrement counter or Check condition
- Loop done?

If Loop done? is Y, exit loop. If N, repeat loop.

Transfer of Control
Structures - 3

Loop:
REPEAT .... UNTIL

Example:
MOV CL, 10H
REPEAT:
.............
.............
.............
.............
DEC CL
JNZ REPEAT

- Initialize counter
- Label indicates where to jump to
- Test condition

4.15

Loop:
WHILE .... DO

Example:
MOV CL, 10H
Again:
.............
.............
.............
.............
DEC CL
JMP AGAIN

- Initialize counter
- Test condition

- Label indicates where to jump to

4.16
Transfer of Control

Examples

Example 1:
Move a block of data starting at memory location LOC1 to LOC2.
The block size is 200 (=C8H) bytes.

```
MOV  CL, C8H   ; Initialize the counter
MOV  ESI, LOC1  ; Source index points to original block location offset
MOV  EDI, LOC2  ; Destination index points to new block location offset
NEXT:  MOV  AL, [ESI]  ; Move bytes
       MOV  [EDI], AL
       INC  ESI  ; Identify (point to) next element
       INC  EDI
       DEC  CL  ; Decrement the counter
       JNZ  NEXT  ; Check if the loop is done
...  
```

Example 2:
A block of data is located at memory location LOC1.
The block size is 40 (= 28H) bytes. Perform:
1 x (byte #1) + 2 x (byte #2) + 3 x (byte#3) + … + 40 x (byte #40).
Assume they are signed numbers!

```
MOV  DX, 0000H  ; Initialize result register
MOV  CL, 01H  ; Initialize the counter
MOV  ESI, LOC1  ; Source index points to original block
NEXT:  MOV  AL, [ESI]  ; Move byte to AL
       IMUL  CL  ; Multiplication
       ADD  DX, AX  ; Addition
       INC  ESI  ; Identify (point to) next element
       INC  CL  ; Decrement the counter
       CMP  CL, 29H  ; Compare with block-size + 1
       JNZ  NEXT  ; Check if the loop is done
...  
```
## Transfer of Control

### Loop and Loop-handling Instructions

<table>
<thead>
<tr>
<th>Loop and Loop-handling:</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LOOP</td>
<td>Loop</td>
<td>LOOP Short-label</td>
</tr>
<tr>
<td></td>
<td>(CX) (\leftarrow) (CX) - 1</td>
<td>Jump is initiated to location defined by short-label if (CX) (\neq) 0; otherwise, execute next sequential instruction</td>
</tr>
<tr>
<td>LOOPE/LOOPZ</td>
<td>Loop while equal/Loop while zero</td>
<td>LOOPE/LOOPZ Short-label</td>
</tr>
<tr>
<td></td>
<td>(CX) (\leftarrow) (CX) - 1</td>
<td>Jump is initiated to location defined by short-label if (CX) (\neq) 0 and (ZF)=1; otherwise, execute next sequential instruction</td>
</tr>
<tr>
<td>LOOPNE/LOOPNZ</td>
<td>Loop while not equal/Loop while not zero</td>
<td>LOOPNE / LOOPNZ Short-label</td>
</tr>
<tr>
<td></td>
<td>(CX) (\leftarrow) (CX) - 1</td>
<td>Jump is initiated to location defined by short-label if (CX) (\neq) 0 and (ZF)=0; otherwise, execute next sequential instruction</td>
</tr>
</tbody>
</table>

### Transfer of Control

#### Examples

**Example 1b:** Move a block of data starting at memory location LOC1 to LOC2. The block size is 200 (=C8H) bytes.

```assembly
MOV CX, 00C8H  ; Initialize the counter
MOV ESI, LOC1  ; Source index points to original location offset
MOV EDI, LOC2  ; Destination index points to new block location offset
NEXT: MOV AL, [ESI]  ; Move bytes
       MOV [EDI], AL
       INC ESI  ; Identify (point to) next element
       INC EDI
       LOOP NEXT  ; Includes a DEC CX and CX =0 flag
...```

4.20
Transfer of Control

Examples

Example 3: One wants to negate a sequence (block) of bytes starting @ LOC1. The block-size is unknown, but the end of the block is indicated by a zero byte. The program must also not negate more than 200 bytes.

Approach:

1) Negate a sequence of bytes: use the 'neg' instruction
2) Starting at address LOC1: move LOC1 to the index pointer
3) End-of-block is zero: check ZF after negate instruction, it should be ZF = 1
4) No more than 200 bytes: use the CX register as a counter
5) Finally, the program needs to continue negating bytes while ZF = 0 and (CX) ≠ 0, thus we can use the LOOPNZ instruction

MOV ESI, LOC1 ; Source index points to original block offset
DEC ESI
MOV CX, 00C8H ; Max number of negations is 200
NEXT:
INC ESI
NEG [ESI] ; Negate bytes
LOOPNZ NEXT ; Includes a DEC CX and (CX) = 0 flag
...
Transfer of Control

Subroutines

- **Subroutine:** a segment of the program that can be called from the main program. When called the microprocessor will stop executing the main program temporarily to execute the subroutine. After the subroutine is executed the microprocessor will return to executing the main program.

- Upon entering a subroutine the contents of certain registers may have to be saved. The mechanism for this is to PUSH these registers onto the stack. Don’t forget to POP the registers again before returning to the main program!

**Operand can be:**

- Near-proc,
- Memptr16,
- Regptr16,
- Far-proc,
- Memptr32.

**Transfer of Control**

**Subroutines**

**Example - 1**

**Problem:** Write a subroutine that subtracts two double-word integers, stored at memory locations 0AC01 and 0AC09, respectively. Note that (DS) = 0A00 and that the AX and BX registers contain important information for the main program. The result must be stored in CX (low-byte) and DX (high-byte).

```
CALL QADD
...........
QADD
PUSH AX
PUSH BX
MOV CX,[0C01]
MOV BX,[0C09]
SUB CX,BX
MOV DX,[0C03]
MOV BX,[0C0B]
SBB DX,BX
POP BX
POP AX
RET
```

: Temporarily store the contents of AX and BX onto the stack

: Subtract with borrow

: Retrieve the contents of AX and BX onto the stack
**Subroutines**

*Example - 2*

**Problem:** Write a subroutine that implements the following equation:

\[ s = s_0 + v_0 t \]

\( s_0 \) and \( v_0 \) are signed bytes stored in memory locations 0A0B1 and 0A0B2, respectively. (DS) = 0A00. \( t \) is stored in DL and the result should be put in DX.

CALL KINE
...........
...........
KINE: MOV AL,DL ; Multiplication always uses AL !
MOV CL,[00B2]
IMUL CL ; Compute the \( v_0 t \) term
MOV DX,AX
MOV AL,[00B1] ; [00B1] contains a signed byte
CBW ; Perform a sign-extension
ADD DX, AX
RET

**Subroutines**

*Passing of Parameters*

- Data is exchanged between the main program and the subroutine in the following ways:
  - Through *registers*,
  - Through *memory locations*,
  - Through a *pointer* pointing to a list or table in memory (list-size is required also).
String Operations

String: Series/Block of Consecutive Words or Bytes

<table>
<thead>
<tr>
<th>Loop and Loop-handling:</th>
<th>MOVSB/MOVSW</th>
<th>CMPSB/CMPSW</th>
<th>CMPS</th>
<th>CMPSB/CMPSW</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV5</td>
<td>Move string</td>
<td>MOV5</td>
<td>Compare string</td>
<td>Compare string</td>
</tr>
<tr>
<td>(ES)+(DI) ← ((DS)+(SI)), (SI) ← (SI) ± 1 or 2, (DI) ← (DI) ± 1 or 2</td>
<td>None</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMPS</td>
<td>Compare string</td>
<td>CMPSB/CMPSW</td>
<td>CMPS</td>
<td>CMPSB/CMPSW</td>
</tr>
<tr>
<td>(SI) ← (SI) ± 1 or 2, (DI) ← (DI) ± 1 or 2</td>
<td>OF,SF,ZF,AF,PF,CF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCAS</td>
<td>Compare string</td>
<td>SCASB/SCASW</td>
<td>SCAS</td>
<td>SCASB/SCASW</td>
</tr>
<tr>
<td>(SI) ← (SI) ± 1 or 2, (DI) ← (DI) ± 1 or 2</td>
<td>OF,SF,ZF,AF,PF,CF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LODS</td>
<td>Load string</td>
<td>LODSB/LODSW</td>
<td>LODS</td>
<td>LODSB/LODSW</td>
</tr>
<tr>
<td>(AL or AX) ← (DS)+(SI), (SI) ← (SI) ± 1 or 2</td>
<td>None</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STOS</td>
<td>Store string</td>
<td>STOSB/STOSW</td>
<td>STOS</td>
<td>STOSB/STOSW</td>
</tr>
<tr>
<td>((ES)+(DI)) ← (AL or AX) ± 1 or 2, (DI) ← (DI) ± 1 or 2</td>
<td>None</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data direction (DF = 0: auto-increment, DF=1: auto-decrement):

| CLD | Clear DF | CLD | (DF) ← 0 | DF |
| SLD | Set DF | SLD | (DF) ← 1 | DF |

Prefixes:

<table>
<thead>
<tr>
<th>Prefixes</th>
<th>Use with MOV5 and STOS</th>
<th>Repeat while not end of string CX ≠ 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>REP</td>
<td>Use with CMPS and SCAS</td>
<td>Repeat while not end of string and strings are equal, CX ≠ 0 and ZF=1</td>
</tr>
<tr>
<td>REPE/REPNZ</td>
<td>Use with CMPS and SCAS</td>
<td>Repeat while not end of string and strings are not equal, CX ≠ 0 and ZF=0</td>
</tr>
</tbody>
</table>
Transfer of Control
Examples

Example 1c: Move a block of data starting at memory location 0C200 to 0C600. The block size is 200 (=C8H) bytes.

```
MOV DS, 0C00 ; Initialize the data segment pointer
MOV ES, 0C00
MOV CX, 00C8H ; Initialize the counter
MOV SI, 0200 ; Source index points to series origin
MOV DI, 0600 ; Destination index points to new series
CLD
NEXT: MOVSB ; Move bytes
LOOP NEXT
HLT
```

Transfer of Control
Examples

Example 1d: Move a block of data starting at memory location 0C200 to 0C600. The block size is 200 (=C8H) bytes.

```
MOV DS, 0C00; Initialize the data segment pointer
MOV ES, 0C00
MOV CX, 00C8H ; Initialize the counter
MOV SI, 0200 ; Source index points to series origin
MOV DI, 0600 ; Destination index points to new series
CLD
REPMOVSB
HLT
```
Transfer of Control

Examples

Example 4: Clear a block of data starting at memory location 0C200. The block size is 200 (=C8H) bytes.

- MOV ES, 0C00H ; Initialize the data segment pointer
- MOV AL, 00H ; Initialization value
- MOV CX, 00C8H ; Initialize the counter
- MOV DI, 0200 ; Source index points to series origin
- CLD
- REPSTOSB
- HLT