**Lecture 6: Machine Code**

- How to do Homework 2!!!

**Homework 2**

- Two parts:
  - Part 1: Use Debug to enter and run a simple machine code program
    - current input data into 2's complement hex
    - enter data at the correct address
    - enter program at the correct address
    - run the program
  - Part 2: Write a simple machine code program, given pseudo-code
    - these instructions should be similar to those in the Part 1 problem.
    - enter and run the resulting program.

---

**Part I - Example Program**

Given below is a machine code program that calculates the sum of all the words in a given range of addresses in memory. The instructions expect that the lower bound of this range is specified in the BX register and the upper bound in the DX register. DX holds the offset of the beginning of the data segment (DS). DS holds the offset of the last data element from the beginning of the data segment. The sum gets stored in AX. The first 4 hex digits given on each line below represent the offset of the instruction from the beginning of the code segment. The digits after the dash are English explanations of the instructions.

```asm
0000 - 2BC0  subtract AX from itself (to make it 0)
0002 - 0307  add the word pointed to by BX to AX
0004 - 83C302  add 2 to BX (to point to the next word)
0007 - 8BD3  compare BX to DX
             (compare sets internal flags that are used by subsequent jump instructions)
0009 - 7DFF  if DX < BX, then jump back to the instruction at 0002
000B - B8004C  this instruction and the next one return control to DOS
000E - CD21
```

---

**Instruction Formats for HW2**

- jump format – jumping from one location in the program to another
- indirect addressing – the source operand is retrieved indirectly, i.e. the operand is at the memory location pointed to by BX
- register to register format – two operands, both are registers
- immediate format – one operand is a constant

---

**immediate format**

**ITR (immediate to register) format**

For two operand instructions in which one of the operands is a specified constant (Example: INC DS instructions above), its general format is:

<table>
<thead>
<tr>
<th>Opcode</th>
<th>op</th>
<th>reg</th>
<th>imm</th>
<th>immediate data</th>
</tr>
</thead>
<tbody>
<tr>
<td>part 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>part 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The opcode for the add instruction in ITR format is 0008H (part 1) and 00H (part 2).

The opcode for the subtract instruction in ITR format is 0009H (part 1) and 00H (part 2).

The opcode for the compare instruction in ITR format is 000DH (part 1) and 11H (part 2).

---

**Let's start by looking at our program. Which instructions are immediate form?**

```asm
0000 - 2BC0  subtract AX from itself (to make it 0)
0002 - 0307  add the word pointed to by BX to AX
0004 - 83C302  add 2 to BX (to point to the next word)
0007 - 8BD3  compare BX to DX
             (compare sets internal flags that are used by subsequent jump instructions)
0009 - 7DFF  if DX < BX, then jump back to the instruction at 0002
000B - B8004C  this instruction and the next one return control to DOS
000E - CD21
```

(actually, we use the latter, but there will always look the same so we won't worry about them now)
Register to register

RTT (register to register) format

For two operand instructions in which both the source and destination operands are registers. (Example: 2BC0
instruction above.) Its general format is:

<table>
<thead>
<tr>
<th>opcode</th>
<th>&amp;</th>
<th>mod</th>
<th>dest reg</th>
<th>src reg</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>1</td>
<td>0</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>0x</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The opcode for the move instruction in RTT format is 000100.
The opcode for the compare instruction in RTT format is 001110.
The opcode for the subtract instruction in RTT format is 001010.


cmp : Register/memory and register

• d = 1, “to” register, d = 0, “from” register
• w = 1 – 16 bit registers (AX, BX, … not AL, BL)
• mod = 11 – src is treated as register field (reg)
• src = 011 = BX (source)

jump format

A two-byte instruction. The first byte designates the condition on which to jump. (Example: 7D =
jump if greater than or equal, in the jump instruction given in the example program. The
opcode 7F means jump if greater than.) The second byte (interpreted as an 8-bit two’s
complement integer) gives the displacement of the jump from the current value of the IP.

• Lets start by looking at our program again. Which instructions are
  register to register form? (look for two registers and no instruction)

JBD3 compare BX to DX
0011 10 11 11 00 11
0000 00 00 00 00 00 00

CMP : Register/memory and register

• d = 1, “to” register, d = 0, “from” register
• w = 1 – 16 bit registers (AX, BX, … not AL, BL)
• mod = 11 – src is treated as register field (reg)
• src = 011 = BX (source)
• mod = 11 – src is treated as register field (reg)
• src = 011 = BX (source)
• wt = 011 = BX (source)

jump format

A two-byte instruction. The first byte designates the condition on which to jump. (Example: 7D =
jump if greater than or equal, in the jump instruction given in the example program. The
opcode 7F means jump if greater than.) The second byte (interpreted as an 8-bit two’s
complement integer) gives the displacement of the jump from the current value of the IP.

• Lets start by looking at our program again. Which instructions are
  in jump form? (look for the word jump)

JBD3 compare BX to DX
0011 10 11 11 00 11
0000 00 00 00 00 00 00

CMP : Register/memory and register

• d = 1, “to” register, d = 0, “from” register
• w = 1 – 16 bit registers (AX, BX, … not AL, BL)
• mod = 11 – src is treated as register field (reg)
• src = 011 = BX (source)
• mod = 11 – src is treated as register field (reg)
• src = 011 = BX (source)
if DX >= BX, then jump back to the instruction at 0002

0111 1101 1111 0111

-> JNE/LJE Jump on not less/Jump greater or equal
+ (we compared DX and BX in the previous instruction and the jump uses the result of the comparison)
+ displacement gives the distance to add to the IP in 2’s complement. In this case it’s 1111 0001 which is a negative number?
1111 0111 -> 0010 1000 + 1 = 0001 0001 = 9
> jump back 9 memory locations

so where are we now? the jump instruction is at 9. But 9 – 9 = 0, not two!
reason: the IP is pointing to the next instruction, which is at 000B. 000B – 0009 = 0002

indirect addressing

The move instruction in your program will use indirect addressing to specify the source operand (i.e. the operand will be at the memory location pointed to by BX). The opposite for a move instruction that uses indirectness is 00010; it fits into the RT6 format given earlier, with mod bits = 00. (Example: 0307 instruction)

0307 add the word pointed to by BX to AX

0000 0011 0001 0111

-> ADD reg/memory with register to refer

*d = 1, reg field holds destination
*w = 1 - 10 bit registers (AX, BX, ... not AL, BL)
+mod = 00 - there are no displacement fields in the instruction
+reg = 000 = AX (dest)
+rmn = 111, EA = (BX) + Dmp
EA = (BX) + Dmp
+EA = effective address - the address of the word being added to AX
+(BX) = contents of BX
+Dmp - an additional displacement field, 0 in this instruction (mod = 00)

more on mod and r/m

• If mod = 11, then r/m is treated as a REG field. This means, you look up the r/m contents on the REG table.
• Otherwise, mod indicates if a displacement is included in the instruction.
• What’s a displacement? Part of the effective address. You’ll see more on displacements when we cover addressing modes.
• So mod is NOT 11, now what do we do?
another example

?? move the word pointed to by BX to DX
look for it in the instruction set list

MOV reg/memory to/from register
(in the data transfer section)

- d = 1 well, we’re moving to DX, a register. So
d = 1
- w = 1 DX is a 16-bit register so w = 1
- mod = 0 well, we are not copying data from a
register. Instead, we are copying data from a
location pointed to by a register.
- reg = 1 well, we are moving the data into DX.
  So, look up the code for DX – 010.
- r/m = 111 there are a lot of choices?

?? move the word pointed to by BX to DX

MOV reg/memory to/from register
(in the data transfer section)

r/m = 000, EA = (BX) + (SI) + DISP
r/m = 001, EA = (BX) + (DI) + DISP
r/m = 010, EA = (BP) + (SI) + DISP
r/m = 011, EA = (BP) + (DI) + DISP
r/m = 100, EA = (SI) + DISP
r/m = 101, EA = (DI) + DISP
r/m = 110, EA = (BP) + (SI) + DISP (exception)
r/m = 111, EA = (BX) + DISP

So which is it? Well, let’s eliminate any that use
registers that are not in our instruction:
SI?
DI?
BP?
this leaves r/m = 111, EA = (BX) + DISP

Entering Data

• You’ll need to do the following:
  - Convert your data into hex.
  Negative numbers are represented
  in 2’s complement.
  - Enter your data into memory at
  the address specified in the
  assignment.
  - Remember, each integer will take
  one word of storage (16 bits) and
  the bytes are stored in reverse
  order!

Entering data example

- Data: 26, 14, -92
- Address for data: 1C554H
  (these are different from your
  assignment?)
- Convert the data:
  26 = 001A, 14 = 000E, -92 = FF44
  (negative numbers are in 2’s complement)
- Set the address: 1C554H
  - data address will an offset from DS
    (data segment register)
  - DS = 1C55H, offset = 4h
    (EA = 1C55H + 4 = 1C554H)

Entering Data (cont.)

• So, to enter the data at 1C554h
  - set DS = 1C55h
  - specify an offset of 4 when
    entering data in Debug
    (w = 4)
  - enter each byte of data,
    remembering that for 16 bit
    values they are stored low byte,
    then high byte
  - 1A 00 0E 00 A4 FF
### Entering the Program

- You’re given the machine code for the program in part 1.
- You’ll need to put it at the correct address.
- Address for program (different from your homework):
  - \(1774\)h
- The code address will be an offset from CS (code segment register)
- CS = \(1774\)h, offset = \(\text{Ch}\).
- So you’ll set the CS register. Then use the "e CS:Cr" command to enter the code.
- You’ll also need to set the IP to 000Ch.

### From the assignment:

The code expects that the lower bound of this range is specified in the BX register and the upper bound in the DX register. (BX holds the offset of the beginning of the data to be summed from the beginning of the data segment (DS). DX holds the offset of the last data element from the beginning of the data segment.)

### Table:

<table>
<thead>
<tr>
<th>EA</th>
<th>Data</th>
<th>Offset from DS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1C554</td>
<td>IA 0004</td>
<td></td>
</tr>
<tr>
<td>1C555</td>
<td>00 0005</td>
<td></td>
</tr>
<tr>
<td>1C556</td>
<td>00 0006</td>
<td></td>
</tr>
<tr>
<td>1C557</td>
<td>00 0007</td>
<td></td>
</tr>
<tr>
<td>1C558</td>
<td>08 0008</td>
<td></td>
</tr>
<tr>
<td>1C559</td>
<td>FF 0009</td>
<td></td>
</tr>
</tbody>
</table>

### Diagram:

- **Offset from CS**: 1C55
- **IP**: 000E
- **AX**: 0000
- **BX**: 0004
- **DX**: 0002
- **CS**: 1774
- **DS**: 000C

While executing (before):
- Subtract AX from itself (to make it 0).

After:
- Subtract AX from itself (to make it 0).
At the end of the program, `DX` points to the last record, `AX` to the next word.

- After: `DX` points to the last record, `AX` to the next word.
- Data: `AX` points to the next word.
- Offsets from `DS` and `CS`:

  - `AX` = `AX`
  - `DX` points to the last record

- If `DX >= BX`, then jump.

To subtract 9:

- Subtract 9: `AX` points to the next word.
- `IP` points to the next word.

- The `DS` field is not used, but it is described in the text.

- The `CS` field is not used, but it is described in the text.

- The `off` field is not used, but it is described in the text.

- The `IP` field is not used, but it is described in the text.

- The `AX` field is not used, but it is described in the text.

- The `DX` field is not used, but it is described in the text.

- The `offs` field is not used, but it is described in the text.

- The `cs` field is not used, but it is described in the text.

- The `ds` field is not used, but it is described in the text.

- The `ip` field is not used, but it is described in the text.

- The `ax` field is not used, but it is described in the text.

- The `dx` field is not used, but it is described in the text.

- The `offs` field is not used, but it is described in the text.

- The `cs` field is not used, but it is described in the text.

- The `ds` field is not used, but it is described in the text.

- The `ip` field is not used, but it is described in the text.

- The `ax` field is not used, but it is described in the text.

- The `dx` field is not used, but it is described in the text.

- The `offs` field is not used, but it is described in the text.

- The `cs` field is not used, but it is described in the text.

- The `ds` field is not used, but it is described in the text.

- The `ip` field is not used, but it is described in the text.

- The `ax` field is not used, but it is described in the text.

- The `dx` field is not used, but it is described in the text.

- The `offs` field is not used, but it is described in the text.

- The `cs` field is not used, but it is described in the text.

- The `ds` field is not used, but it is described in the text.

- The `ip` field is not used, but it is described in the text.

- The `ax` field is not used, but it is described in the text.

- The `dx` field is not used, but it is described in the text.

- The `offs` field is not used, but it is described in the text.

- The `cs` field is not used, but it is described in the text.

- The `ds` field is not used, but it is described in the text.

- The `ip` field is not used, but it is described in the text.

- The `ax` field is not used, but it is described in the text.

- The `dx` field is not used, but it is described in the text.

- The `offs` field is not used, but it is described in the text.

- The `cs` field is not used, but it is described in the text.

- The `ds` field is not used, but it is described in the text.

- The `ip` field is not used, but it is described in the text.

- The `ax` field is not used, but it is described in the text.

- The `dx` field is not used, but it is described in the text.

- The `offs` field is not used, but it is described in the text.

- The `cs` field is not used, but it is described in the text.

- The `ds` field is not used, but it is described in the text.

- The `ip` field is not used, but it is described in the text.

- The `ax` field is not used, but it is described in the text.

- The `dx` field is not used, but it is described in the text.

- The `offs` field is not used, but it is described in the text.

- The `cs` field is not used, but it is described in the text.

- The `ds` field is not used, but it is described in the text.

- The `ip` field is not used, but it is described in the text.

- The `ax` field is not used, but it is described in the text.

- The `dx` field is not used, but it is described in the text.

- The `offs` field is not used, but it is described in the text.

- The `cs` field is not used, but it is described in the text.

- The `ds` field is not used, but it is described in the text.

- The `ip` field is not used, but it is described in the text.

- The `ax` field is not used, but it is described in the text.

- The `dx` field is not used, but it is described in the text.

- The `offs` field is not used, but it is described in the text.

- The `cs` field is not used, but it is described in the text.

- The `ds` field is not used, but it is described in the text.

- The `ip` field is not used, but it is described in the text.

- The `ax` field is not used, but it is described in the text.

- The `dx` field is not used, but it is described in the text.

- The `offs` field is not used, but it is described in the text.

- The `cs` field is not used, but it is described in the text.

- The `ds` field is not used, but it is described in the text.

- The `ip` field is not used, but it is described in the text.

- The `ax` field is not used, but it is described in the text.

- The `dx` field is not used, but it is described in the text.

- The `offs` field is not used, but it is described in the text.

- The `cs` field is not used, but it is described in the text.

- The `ds` field is not used, but it is described in the text.

- The `ip` field is not used, but it is described in the text.

- The `ax` field is not used, but it is described in the text.

- The `dx` field is not used, but it is described in the text.

- The `offs` field is not used, but it is described in the text.

- The `cs` field is not used, but it is described in the text.

- The `ds` field is not used, but it is described in the text.

- The `ip` field is not used, but it is described in the text.

- The `ax` field is not used, but it is described in the text.

- The `dx` field is not used, but it is described in the text.

- The `offs` field is not used, but it is described in the text.

- The `cs` field is not used, but it is described in the text.

- The `ds` field is not used, but it is described in the text.

- The `ip` field is not used, but it is described in the text.

- The `ax` field is not used, but it is described in the text.

- The `dx` field is not used, but it is described in the text.

- The `offs` field is not used, but it is described in the text.

- The `cs` field is not used, but it is described in the text.

- The `ds` field is not used, but it is described in the text.

- The `ip` field is not used, but it is described in the text.

- The `ax` field is not used, but it is described in the text.

- The `dx` field is not used, but it is described in the text.

- The `offs` field is not used, but it is described in the text.

- The `cs` field is not used, but it is described in the text.

- The `ds` field is not used, but it is described in the text.

- The `ip` field is not used, but it is described in the text.

- The `ax` field is not used, but it is described in the text.

- The `dx` field is not used, but it is described in the text.

- The `offs` field is not used, but it is described in the text.

- The `cs` field is not used, but it is described in the text.

- The `ds` field is not used, but it is described in the text.

- The `ip` field is not used, but it is described in the text.

- The `ax` field is not used, but it is described in the text.

- The `dx` field is not used, but it is described in the text.

- The `offs` field is not used, but it is described in the text.

- The `cs` field is not used, but it is described in the text.

- The `ds` field is not used, but it is described in the text.

- The `ip` field is not used, but it is described in the text.

- The `ax` field is not used, but it is described in the text.

- The `dx` field is not used, but it is described in the text.

- The `offs` field is not used, but it is described in the text.

- The `cs` field is not used, but it is described in the text.

- The `ds` field is not used, but it is described in the text.

- The `ip` field is not used, but it is described in the text.

- The `ax` field is not used, but it is described in the text.

- The `dx` field is not used, but it is described in the text.

- The `offs` field is not used, but it is described in the text.

- The `cs` field is not used, but it is described in the text.

- The `ds` field is not used, but it is described in the text.

- The `ip` field is not used, but it is described in the text.

- The `ax` field is not used, but it is described in the text.

- The `dx` field is not used, but it is described in the text.

- The `offs` field is not used, but it is described in the text.

- The `cs` field is not used, but it is described in the text.

- The `ds` field is not used, but it is described in the text.

- The `ip` field is not used, but it is described in the text.
Part 2?

- Very similar to part one, except you need to figure out the machine code.
- Most of these instructions are similar to those in part 1 and can be created with minor modifications to the part 1 instructions.
- Read the assignment carefully to make sure you are putting the program and data in the correct locations!