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
    - convert 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 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.) 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 the machine code instructions. To the right are English explanations of the instructions.

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 - 3BD3	compare BX to DX (compare sets internal flags that are used by subsequent jump instructions)
0009 - 7DF7	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: 83C302 instruction above). Its general format is:

opcode	s	w	mod	opcode	reg	immediate data
part 1				part 2		
	1	1	11			
23-18	17	16	15-14	13-11	10-8	7-0

The opcode for the add instruction in ITR format is 100000 (part 1) and 000 (part 2).
The opcode for the subtract instruction in ITR format is 100000 (part 1) and 101 (part 2).
The opcode for the compare instruction in ITR format is 100000 (part 1) and 111 (part 2).

- Lets start by looking at our program. Which instructions are immediate form?

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 - 3BD3	compare BX to DX (compare sets internal flags that are used by subsequent jump instructions)
0009 - 7DF7	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, so are the last two, but those will always look the same so we won't worry about them now)

83C302 **add 2 to BX (to point to the next word)**

10000 0 11 11 000 011 0000 0010

1 0 0 0 0 s w	mod 0 0 0 r/m	data	data if sw=01

→ ADD: Immediate to register/memory

- sw = 11 – 8 bits of immediate data (sign extended to 16 when used)
- mod = 11 – r/m is treated as register field (reg) and specifies which register the instruction uses
- r/m as register? Look on handout where it says REG:
  - 000 = AX
  - 001 = CX
  - 010 = DX
  - 011 = BX
  - 100 = SP…. etc.
  - r/m = 011 so our register = BX
- what's left? Our one byte (8 bits) of data:
  - 0000 0010 = 2

register to register format

RTR (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:

opcode	d	w	mod	dest reg	src reg
					(r/m)
	1	1	11		
15 - 10	9	8	7-6	5-3	2-0

The opcode for the **move** instruction in RTR format is 100010.
The opcode for the **compare** instruction in RTR format is 001110.
The opcode for the **subtract** instruction in RTR format is 001010.

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

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 - 3BD3	**compare BX to DX** (compare sets internal flags that are used by subsequent jump instructions)
0009 - 7DF7	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	

3BD3 compare BX to DX

0011 10 11 11 01 0011

0 0 1 1 1 0 d w mod reg r/m

→CMP: Register/memory and register

- d = 1, "to" register, d = 0, "from" register
  in our case, d = 1 so the reg field gives the "to" register
- w = 1 – 16 bit registers (AX, BX,…not AL, BL)
- mod = 11 – r/m is treated as register field (reg)
- reg = 010 = DX (dest)
- r/m = 011 = BX (source)

(CMP does an "implied subtract" of dest ("to") - source ("from"), and sets the appropriate flags)

jump format

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)

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 - 3BD3	compare BX to DX (compare sets internal flags that are used by subsequent jump instructions)
0009 - 7DF7	**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	

7DF7 if DX >= BX, then jump back to the instruction at 0002

0111 1101 1111 0111

0 1 1 1 1 1 0 1	disp

→ JNL/JGE: 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 0111 which is a negative number!

1111 0111 -> 0000 1000 + 1 = 0000 1001 = 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

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 - 3BD3	compare BX to DX (compare sets internal flags that are used by subsequent jump instructions)
0009 - 7DF7	**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	

when executing the instruction at 0009, the IP is pointing to the *next* instruction to be executed: the instruction at 000B.

So, 000B (the address in IP) – 0009 (the amount to jump back (which just happens to be equal to the address of the current instruction) = 0002 (the address of the *next* instruction to execute.

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 opcode for a move instruction that uses indirection is 100010; it fits into the RTR format given earlier, with mod bits = 00. (Example: 0307 instruction).

- Lets start by looking at our program again. Which instructions use indirect addressing? (look for instructions where the comments say "pointed to"

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 - 3BD3	compare BX to DX (compare sets internal flags that are used by subsequent jump instructions)
0009 - 7DF7	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	

0307 add the word pointed to by BX to AX

0000 0011 0000 0111

0 0 0 0 0 0 d w	mod reg r/m

→ADD: reg/memory with register to either

- d = 1, reg field holds destination
- w = 1 – 16 bit registers (AX, BX,…not AL, BL)
- mod = 00 – there are no displacement fields in the instruction.
- reg = 000 = AX (dest)
- r/m = 111, EA = (BX) + Disp

EA = (BX) + Disp?

- EA = effective address – the address of the word being added to AX
- (BX) = contents of BX
- Disp – 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

1 0 0 0 1 0 d w	mod reg r/m

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

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

?? move the word pointed to by BX to DX

1 0 0 0 1 0 d w	mod reg r/m

→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) + DISP (w/exception)
r/m = 111, EA = (BX) + DISP

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

?? move the word pointed to by BX to DX

1 0 0 0 1 0 d w	mod reg r/m

d = 1 because desination is a register
w = 1 because it's a 16-bit instruction
mod = 00 because there is no displacement
reg = 010 for DX
r/m = 111 for indirect addressing where the address is stored in BX.

100010 1 1 00 010 111
1000 1011 0001 0111 (spacing for convenience)
= 8B17

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 = FFA4

  (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 = 1C550 + 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 `(e ds: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

EA	Data	Offset from DS
1C554	1A	0004
1C555	00	0005
1C556	0E	0006
1C557	00	0007
1C558	A4	0008
1C559	FF	0009

DS: 1C55

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):*
  1774Ch
- The code address will be an offset from CS (code segment register)
- CS = 1774h, offset = Ch.
- So you'll set the CS register. Then use the "e CS:C" command to enter the code.
- You'll also need to set the IP to 000Ch

EA	Code	Offset from CS
1774C	2B	000C
1774D	C0	000D
1774E	03	000E
1774F	07	000F
17750	83	0010
17751	C3	0011
17752	02	0012
17753	3B	0013
17754	D3	0014
17755	7D	0015
17756	F7	0016
17757	B8	0017
17758	00	0018
17759	4C	0019
1775A	CD	001A
1775B	21	001B

CS: 1774

IP: 000C

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.)

EA	Data	Offset from DS
1C554	1A	0004
1C555	00	0005
1C556	0E	0006
1C557	00	0007
1C558	A4	0008
1C559	FF	0009

DS: 1C55

BX:

DX:

Offset from CS	Code	Data	Offset from DS
000C	2B		
000D	C0	1A	0004
000E	03	00	0005
000F	07	0E	0006
0010	83	00	0007
0011	C3	A4	0008
0012	02	FF	0009
0013	3B		
0014	D3		
0015	7D		
0016	F7		
0017	B8		
0018	00		
0019	4C		
001A	CD		
001B	21		

Register	Value
IP	000C
BX	0004
DX	0008
AX	?
CS	1774
DS	1C55

While executing (before):
2BC0 subtract AX from itself (to make it 0)

Offset from CS	Code	Data	Offset from DS
000C	**2B**		
000D	**C0**	1A	0004
000E	03	00	0005
000F	07	0E	0006
0010	83	00	0007
0011	C3	A4	0008
0012	02	FF	0009
0013	3B		
0014	D3		
0015	7D		
0016	F7		
0017	B8		
0018	00		
0019	4C		
001A	CD		
001B	21		

Register	Value
IP	000E
BX	0004
DX	0008
AX	**0000**
CS	1774
DS	1C55

After:
2BC0 subtract AX from itself (to make it 0)

Offset from CS	Code	Data	Offset from DS
000C	2B		
000D	C0	**1A**	0004
000E	**03**	**00**	0005
000F	**07**	0E	0006
0010	83	00	0007
0011	C3	A4	0008
0012	02	FF	0009
0013	3B		
0014	D3		
0015	7D		
0016	F7		
0017	B8		
0018	00		
0019	4C		
001A	CD		
001B	21		

Register	Value
IP	0010
BX	0004
DX	0008
AX	**001A**
CS	1774
DS	1C55

After:
0307 add the word pointed to by BX to AX

Offset from CS	Code	Data	Offset from DS
000C	2B		
000D	C0	1A	0004
000E	03	00	0005
000F	07	0E	0006
0010	**83**	00	0007
0011	**C3**	A4	0008
0012	**02**	FF	0009
0013	3B		
0014	D3		
0015	7D		
0016	F7		
0017	B8		
0018	00		
0019	4C		
001A	CD		
001B	21		

Register	Value
IP	0013
BX	**0006**
DX	0008
AX	001A
CS	1774
DS	1C55

After:
83C302 add 2 to BX (to point to the next word)

Offset from CS	Code
000C	2B
000D	C0
000E	03
000F	07
0010	83
0011	C3
0012	02
0013	**3B**
0014	**D3**
0015	7D
0016	F7
0017	B8
0018	00
0019	4C
001A	CD
001B	21

Data	Offset from DS
1A	0004
00	0005
0E	0006
00	0007
A4	0008
FF	0009

Register	Value
IP	0015
BX	0006
DX	0008
AX	001A

After:
3BD3 compare BX to DX (sets flags)

NV - no overflow
PL – positive
overflow = sign: jump

Offset from CS	Code
000C	2B
000D	C0
000E	03
000F	07
0010	83
0011	C3
0012	02
0013	3B
0014	D3
0015	7D
0016	F7
0017	B8
0018	00
0019	4C
001A	CD
001B	21

Data	Offset from DS
1A	0004
00	0005
0E	0006
00	0007
A4	0008
FF	0009

Register	Value
IP	000E
BX	0006
DX	0008
AX	001A

After:
7DF7 if DX >= BX, then jump back 9 locations

overflow = sign: jump
old IP = 0017,
subtract 9: 000E

Offset from CS	Code
000C	2B
000D	C0
000E	**03**
000F	**07**
0010	83
0011	C3
0012	02
0013	3B
0014	D3
0015	7D
0016	F7
0017	B8
0018	00
0019	4C
001A	CD
001B	21

Data	Offset from DS
1A	0004
00	0005
0E	0006
00	0007
A4	0008
FF	0009

Register	Value
IP	0010
BX	0006
DX	0008
AX	**0028**

After:
0307 add the word pointed to by BX to AX

1A + 0E
= 28h

Offset from CS	Code
000C	2B
000D	C0
000E	03
000F	07
0010	**83**
0011	**C3**
0012	**02**
0013	3B
0014	D3
0015	7D
0016	F7
0017	B8
0018	00
0019	4C
001A	CD
001B	21

Data	Offset from DS
1A	0004
00	0005
0E	0006
00	0007
A4	0008
FF	0009

Register	Value
IP	0013
BX	**0008**
DX	0008
AX	0028

After:
83C302 add 2 to BX (to point to the next word)

Offset from CS	Code
000C	2B
000D	C0
000E	03
000F	07
0010	83
0011	C3
0012	02
0013	3B
0014	D3
0015	**7D**
0016	**F7**
0017	B8
0018	00
0019	4C
001A	CD
001B	21

Data	Offset from DS
1A	0004
00	0005
0E	0006
00	0007
A4	0008
FF	0009

Register	Value
IP	000E
BX	0008
DX	0008
AX	0028

After:
7DF7 if DX >= BX, then jump back 9 locations

overflow = sign: jump
old IP = 0017,
subtract 9: 000E

Offset from CS	Code
000C	2B
000D	C0
000E	**03**
000F	**07**
0010	83
0011	C3
0012	02
0013	3B
0014	D3
0015	7D
0016	F7
0017	B8
0018	00
0019	4C
001A	CD
001B	21

Data	Offset from DS
1A	0004
00	0005
0E	0006
00	0007
A4	0008
FF	0009

Register	Value
IP	0010
BX	0008
DX	0008
AX	**FFCC**

After:
0307 add the word pointed to by BX to AX

28h + FFA4
= FFCCh

Offset from CS	Code
000C	2B
000D	C0
000E	03
000F	07
0010	**83**
0011	**C3**
0012	**02**
0013	3B
0014	D3
0015	7D
0016	F7
0017	B8
0018	00
0019	4C
001A	CD
001B	21

Data	Offset from DS
1A	0004
00	0005
0E	0006
00	0007
A4	0008
FF	0009

Register	Value
IP	0013
BX	**000A**
DX	0008
AX	FFCC

After:
83C302 add 2 to BX (to point to the next word)

Offset from CS	Code
000C	2B
000D	C0
000E	03
000F	07
0010	83
0011	C3
0012	02
0013	**3B**
0014	**D3**
0015	7D
0016	F7
0017	B8
0018	00
0019	4C
001A	CD
001B	21

Data	Offset from DS
1A	0004
00	0005
0E	0006
00	0007
A4	0008
FF	0009

Register	Value
IP	0015
BX	000A
DX	0008
AX	FFCC

After:
3BD3 compare BX to DX (sets flags)

NV – no overflow
NG - negative
overflow /= sign:
no jump!

Offset from CS	Code
000C	2B
000D	C0
000E	03
000F	07
0010	83
0011	C3
0012	02
0013	3B
0014	D3
0015	**7D**
0016	**F7**
0017	B8
0018	00
0019	4C
001A	CD
001B	21

Data	Offset from DS
1A	0004
00	0005
0E	0006
00	0007
A4	0008
FF	0009

IP	000E
BX	000A
DX	0008
AX	FFCC

overflow/ = sign:
do not jump!

After:
7DF7 if DX >= BX, then jump back 9 locations

Before:
B8004C copy 4C00h into AX
(you'll learn about this later!)

Result

- At the end of the program *(but prior to the last two machine instructions)*, AX holds the result: FFCCh
- This is a 2's complement result. As we learned earlier, if a 2's complement number has a 1 in the left-most bit, it is negative.
  - FFCCh = 1111 1111 1100 1100
  - = - 0000 0000 0011 0011 + 1
  - = - 0000 0000 0011 0100
  - = - (2 **5 + 2 **4 + 4)
  - = - 52
- Is this correct?
  - 26 + 14 – 92 = 40 – 92 = -52

(if the left-most bit of your answer was zero, you would be able to simply convert it to decimal with no inverting)

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!