EC327 Introduction to Software Engineering – Fall 2015

SIMPLIFIED INTEL ASSEMBLY

(Homework 1)

Introduction

The following is a (greatly simplified) Intel assembly and machine language for a 16-bit computer for use in Homework 1. *Note that your code should start (i.e. first instruction) at address 31 (decimal).

Intel assembly language

Our simplified machine will have six 16-bit general-purpose registers labeled R0 through R5 (the real Intel general purpose registers are 32-bit and have names like EAX , EBX , ECX , EDX , EBP , and ESP) with R6 representing the program counter PC. All target addresses are 12-bits long (meaning that the machine has at most 4096 memory locations).
We will also utilize the following instructions:

halt

    halt execution

---

add Rn, Rm

    register Rn gets the sum of the contents of Rn and Rm

sub Rn, Rm

    register Rn gets the result of contents of Rn minus contents of Rm

inc Rn

    register Rn gets contents of Rn + 1

xor Rn, Rm

    Rn gets the bit-wise exclusive or of Rn and Rm

---

jmp <target>

    jump to label ;

cmp Rn, Rm

    Compares Rn and Rm and sets appropriate flags

je <target>

    jump to label if the last cmp instruction compared equal items

jne <target>

    jump to label if the last cmp instruction compared unequal items

---

mov Rn, num

    put the hexadecimal value *num* into register *Rn*

mov Rn, Rm

    copy data from register Rm into register Rn.

mov [Rn], Rm

    copy data from register Rm into the memory address stored in Rn (Mem[Rn] = Rm)

mov Rn,[Rm]

    copy data from memory address stored in Rm to register Rn (Rn = Mem[Rm])

Notice that only the lowest 12 bits of indirect addresses ([Rn] or [Rm]) are used.

Example

Simple

The following example adds 2 and 2 and stores the result in R0

Instruction	Parameters	Comment
mov	R0,0	*put the number 0 into register R0*
mov	R1,2	*put the number 2 into register R1*
add	R0, R1	*add the contents of R1 to R0; R0 now stores the number 2*
add	R0, R1	*add the contents of R1 to R0; R0 now stores the number 4*

Harder

The following code will fill memory locations 10 to 19 with the numbers 0 through 9 respectively, and then halts:

Address	Instruction	Parameters	Comment
31	mov	R0,0	*R0=0 - the number to put into memory, starts at 0*
32	mov	R1, 10	*R1 = 10, the current memory location, starts at 10*
33	mov	R2, 20	*R2 = 20, last memory location+1 (for performing comparison)*
34 theLoop:	mov	[R1], R0	*memory[R1] = contents of R0*
35	inc	R0	*R0 = R0 + 1*
36	inc	R1	*R1 = R1 + 1*
37	cmp	R1, R2	*is the current memory location (R1) equal to the last memory location? (R2 =20)*
38	jne	theLoop	*if the above comparison shows R1 not equal to 20, jump back to theLoop*
39	halt		*halts execution*

Machine language

The actual translation from assembly in machine language can be fairly complicated, and we will thus use an extremely simplified (and not accurate) alternative. We will encode each assembly language instruction in 16 bits (represented in hex) based on the number of parameters it has:

- The single no-parameter instruction:

Instruction	4-bit code
halt	0000

- Single parameter instructions encode the instruction in the first 4 bits, and the parameter in the last 12 bits. Instruction codes are:

Instruction	4-bit code
inc	0001
jmp	0010
jne	0011
je	0100

The parameter is either a register number (0-6; for inc) or the address of a target (for jumps).

- Two parameter instructions are encoded as a 4-bit instruction, followed by two 6-bit numbers each consisting of 0 to 6 (inclusive) for a register (R0 to R6) **or** 0 to 63 (inclusive) for an integer. Specific instruction codes are:

Instruction	4-bit code
add	0101
sub	0110
xor	0111
cmp	1000
mov Rn, num	1001
mov Rn, Rm	1010
mov [Rn], Rm	1011
mov Rn, [Rm]	1100

Example

The assembly instruction inc R2 would be encoded as the 16-bit sequence:

Instruction	Register
0001	000000000010

In hex, this would be 0x1002.

The instructions in the harder assembly example above could be encoded as:

Memory address	Machine code	Assembly
0-30	0000000000000000 (*Hex:* 0000)	*empty cells*
31	1001 000000 000000 (*Hex:* 9000)	mov R0,0
32	1001 000001 001010 (*Hex:* 904A)	mov R1, 10
33	1001 000010 010100 (*Hex:* 9094)	mov R2, 20
34 theLoop:	1011 000001 000000 (*Hex:* B040)	mov [R1], R0
35	0001 000000000000 (*Hex:* 1000)	inc R0
36	0001 000000000001 (*Hex:* 1001)	inc R1
37	1000 000001 000010 (*Hex:* 8042)	cmp R1, R2
38	0011 000000100010 (*Hex:* 3022)	jne theLoop (address is 34 = 0x22)
39	0000 000000000000 (*Hex:* 0000)	halt