Computer Systems (113) / Architecture (110)

Toy Architecture/Programming - Questions

1 The TOY1 machine features the following instruction with opcode 0110.

IFZER R**n**, **address**

The meaning of this instruction is

IF R**n** = 0 THEN ProgramCounter = **address**

(i) Give the binary machine instruction for IFZER R2, 111H. Write the instruction in hex also.

(ii) Assume this instruction is located at address 80H in memory and that the Program Counter points to this instruction (i.e. the Program Counter = 80H). Show the micro-steps within the CPU and across the address/data buses for this instruction when:

(a) R2=0

(b) R2=55H

2 TOY2 is a 32-bit successor to TOY1. Its instructions include:

Register[N] = Memory [Address]

Memory [Address] = Register[N]

Register[N] = Register[M]

Register[N] = Register[M] + Register[P]

Register[N] = Register[M] - Register[P]

Devise a 32-bit instruction format or formats for these instructions. For your format(s) state

(i) the number of instructions catered for

(ii) the number of registers catered for

(iii) the number of words in memory catered for

Note: There is no single correct format. Each student should design their own format(s).

3 (i) Write a TOY1 assembler program (instructions plus constants) to change the sign of all negative numbers in a list of 32 integers stored consecutively from memory address 1F0H upwards.

Hint: Develop a high-level language solution first.

(ii) Translate you program into binary TOY1 machine instructions, indicating clearly the address of each instruction/constant.

Computer Systems (113) / Architecture (110)

Toy Architecture/Programming - Solutions

1(i)

Recall that the format for this instruction is:

OP	REG	Address
4-bits	2-bits	10-bits

therefore IFZER R2, 111H would translate to

0110	10 01	0001	0001	Binary
6	9	1	1	Hex

1(ii) (a) For R2=0 we have:

PC	080H		080H	Address Bus
0 to Control Bus	0		0	Control Bus
Address Bus	080H		080H	Memory
Control Bus	0		0	Memory
Increment PC	080H		081H	PC=PC+1
Memory [080H	6911H		6911H	Data Bus
Data Bus	6911H		6911H	Instruction Register
Instruction Register	6911H		6911H	Instruction Decoder
Instruction Decoder	6, 2, 111H		6, 2, 111H	Control Unit
Register R2	0		0	ALU Input Register 1
Control Unit to ALU, send **IFZER** opcode			1(ie. True)	ALU Output Register
ALU Output Register	1 (ie. True)		1	Control Unit
Control Unit	111H		111H	PC

1(ii) (b) For R2=55H the last 4 steps above would be replaced by:

Register R2	55H		55H	ALU Input Register 1
Control Unit to ALU send **IFZER** opcode			0(ie. False)	ALU Output Register
ALU Output Register	0(ie. False)		0	Control Unit

2 There are many possible designs. Here's one:

Instruction Format for

Register[N] = Memory [Address]
Memory [Address] = Register[N]

OP	Register[N]	Address
6-bits	6-bits	20-bits

Instruction Format for

Register[N] = Register[M]

OP	Register[N]	Register[M]	Unused
6-bits	6-bits	6-bits	14-bits

Instruction Format for

Register[N] = Register[M] + Register[P]
Register[N] = Register[M] - Register[P]

OP	Register[N]	Register[M]	Register[P]	Unused
6-bits	6-bits	6-bits	6-bits	8-bits

(i) For 6-bit opcode we have 64 instructions (i.e. 2^6)

(ii) 6-bits for the register number gives us 64 registers

(ii) 20-bit address gives us 1M words of memory (word-addressable)
or if memory word size = 32 bits and memory is byte addressable then
20-bit address = 1M bytes = 256Kwords of memory.

Note: This solution describes an architecture whose instructions are of the same size. Some real architectures support variable size instructions. Why?

3.

```
n = 32
addr = 1F0H
loop exit when n <= 0
        if Memory[addr] < 0 then
                Memory[addr] = - Memory[addr]
        end if
        addr = addr+1
        n = n - 1
end loop
```

Register Allocation. We'll use:
Register 1 for *n*
Register 2 for *addr*.
Register 3 will be employed as a temporary register.

Addr	Assembler Instr	Comment	Machine Instruction
0	0	Zero	0000 0000 0000 0000
1	1	One	0000 0000 0000 0001
2	32	No of integers	0000 0000 0010 0000
3	1F0H	Start address	0000 0001 1111 0000
....			
10H	**LOAD** R1, [2H]	n = 32	0001 0100 0000 0010
11H	**LOAD** R2, [3H]	addr =1F0H	0001 1000 0000 0011
12H	**IFZER** R1, 1DH	**loop exit when** n<=0	0110 0100 0001 1101
13H	**IFNEG** R1, 1DH		0111 0100 0001 1101
14H	**LOAD** R3, [R2]	R3 =Memory[addr]	1001 1110 xxxx xxxx
15H	**IFNEG** R3, 17H	**if** R3 < 0	0111 1100 0001 0111
16H	**GOTO** 1AH	**then**	0101 xx00 0001 1010
17H	**LOAD** R3, [0]	R3 = 0	0001 1100 0000 0000
18H	**SUB** R3, [R2]	R3 = -Memory[addr]	1100 1110 xxxx xxxx
19H	**STORE** R3, [R2]	Memory[addr]=R3	1010 1110 xxxx xxxx
		end if	
1AH	**ADD** R2, [1]	addr = addr+1	0011 1000 0000 0001
1BH	**SUB** R1, [1]	n = n-1	0100 0100 0000 0001
1CH	**GOTO** 12H	**end loop**	0101 xx00 0001 0010
1DH	**STOP**		0000 0000 0000 0000

x's are <u>don't care</u> bits i.e. we don't care whether they are 0 or 1.