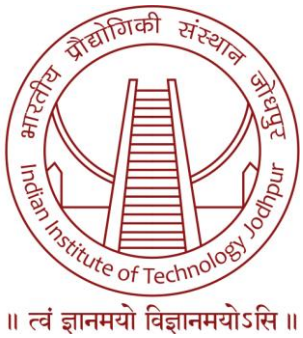
Computer Architecture

CSL7070

Assignment1: ISA and RISC PROGRAMMING

Instructor: Dip Sankar Banerjee



Submitted by: Ajit Kumar

Assignment 1

Q2(a):

ISA basically deals with the interface between software and hardware and Microarchitecture deals with the implementation in hardware. This distinction is important because this allows software to be independent of hardware. Also, as a programmer we do not need to worry about what actually is the underlying architecture. The Optimizations as well as modifications can be made at the microarchitecture level without affecting the legacy software which was developed earlier.

Q2(b):

Compiler does not need to know about microarchitecture because only the knowledge of ISA is sufficient since the compiler will be generating the ISA. But, on the other hand knowledge of microarchitecture is very important because as a programmer we can help the compiler to perform all the optimizations which the compiler will always try to do like register usage, branch prediction, out-of-order execution & loop unrolling.

Q3:

a. Instructions are 32 bits wide: **classified as ISA**

b. Instructions are always executed in order: **classified as microarchitecture**

c. The ALU does not have a subtraction module: **classified as microarchitecture**

d. There is no multiplication instruction: **classified as ISA**

e. The MAR and MDR registers are used for memory reads and writes: **classified as microarchitecture**

f. There are 12 general purpose registers that instructions can use: **classified as ISA**

g. There are three condition codes (n, z, and p) representing the result of the previous instruction: **classified as ISA**

h. It takes 6 cycles to execute a multiplication instruction: **classified as microarchitecture**

i. Instructions are pipelined in four stages: **classified as microarchitecture**

Q4(a):

One use case for a NOP instruction is to insert the delays.

Q4(b):

ADDI x0, x0, 0

Q5(a):

This decision is an ISA design specification.

Q5(b):

Both 2nd and 3rd

Q5(c):

Only 3rd

Q6(1):

X1200+ X3200=X4400

Q6(2):

X0012+X0032=X0044

Q7(1):

Total memory =64MB

=2^26Byte

=2^29 bits

=2^22 128-bit

29 bits

Q7(2):

26 bits

Q7(3):

22 bits

Q8(a):

Zero-address machine:

PUSH B

PUSH C

MUL

PUSH A

ADD

PUSH D

PUSH C

MUL

PUSH E

ADD

PUSH D

SUB

MUL

POP X

Q8(b):

One-address machine:

LOAD C, MUL D, ADD E, STORE T, LOAD D, SUB T, STORE T, LOAD B, MUL C, ADD A, MUL T, STORE X

Q8(c):

Two-address machine:

MOV R1, B

MUL R1, C

Add R1, A

MOV R2, D

MUL R2, C

Add R2, E

Mov R3, D

Sub R3, R2

Mul R1, R3

Q8(d):

Three-address machine:

Mul R1, B, C

Add R2, A, R1

Mul R1, D, C

Add R3, E, R2

Sub R1, D, R3

Mul X, R2, R1

Q10:

Original Ques:

```
MAIN: LA t2,L0

JALR R2
JR L1
ADDI A0, X0, 10

ECALL
L0: ADDI t0,x0,5

RET
L1: ADDI t1,t1,5

RET
```

Updated:

```
MAIN: LA t2, L0     # load L0 value to t2


JALR t2          #jump and link register to t2

JR t2

ADDI a0, x0, 10    # add 10 to x0 and assigning it to a0


ECALL

L0: ADDI t0, x0, 5   # add 5 to x0 and assign it to t0


RET

L1: ADDI t1, t1,5   # add 5 to t1 and assign it to itself


RET
```

Q10(1):

PC	Machine Code	Basic Code	Original Code
0x4	0x01838393	addi x7 x7 24	MAIN: LA t2,L0
0x8	0x000380E7	jalr x1 x7 0	JALR t2
0xc	0x00038067	jalr x0 x7 0	JR t2
0x10	0x00A00513	addi x10 x0 10	ADDI a0, x0, 10
0x14	0x00000073	ecall	ECALL
0x18	0x00500293	addi x5 x0 5	L0: ADDI t0,x0,5
0x1c	0x00008067	jalr x0 x1 0	RET
0x20	0x00530313	addi x6 x6 5	L1: ADDI t1,t1,5
0x24	0x00008067	jalr x0 x1 0	RET

Q10(2):

JR is a pseudo instruction which gets converted to JALR. So, JR means JALR.

Q9(1):

8 bits

Q9(2):

4 General Purpose Registers

Q9(3):

AND R2 0 // setting 0 in R2

ADD R2 R1 //Adding R2 and R1

Q11(1):

because argument register is not initialized t1. It should be initialized to 1 to behaving as expected.

2ans:

```
MAIN:La t3, L2                  #load address of l2 to t3

#A2:    JL2

addi a0, x0, 1              #add 1 to x0 and assigning it to a0

addi a1, x0, 10             #add 10 to x0 and assigning to a1

ecall

L2: ADD t2, t1, x0          #add x0 value to t1 and assigning it to t2

RET
```

3ans:

```
MAIN: LA t3, L2   #load L2 value to t3

A2:  J L2    #jump to l2

addi a1, x0, 10   # add 10 to x0 and assigning it to a1

ecall

L2:

Addi a0, x0, 1   # add 1 to x0 and assigning it to a0

Addi a1, x0,10   # add 10 to x0 and assigning it to a1

Ecall

RET
```

Q12:

Tradeoffs between variable length ISA & fixed instruction length ISA:

- complexity of hardware tradeoff
- Scalability Performance tradeoff
- Size of code and execution speed tradeoff

How do variable length instructions affect the hardware & Software:

- Faster execution, as number of registers will be more
- Better Addressing modes can be provided

Q13(1):

t5 after value is 10

t7 after value is 28

Q13(2):

addi t3 t3 18 #adding 18 to t3 and assigning it to t3 itself

addi t6 t6 –2 #adding –2 to t6 and assigning it to t6 itself

add t5 t5 t1 #adding t5 to 1 and assigning t5 to itself

add s2 t3 t4 // no t7 register present, hence we s2

Q13(3):

Machine Code is as follows:

0x00x012E0E13

180x40xFFEF8F93

0x01CF0F33

0x00730433