Name: Guttu Sai Abhishek, Roll No:180050036

Computer Architecture Theory + Lab (CS 305/341)

Assignment 2: Basics of ISA Due Date: 08/09/20

(Theory Assignment 1)

1. In the layered diagram of a computer system why does
   (a) The OS appear above the Architecture layer
   (b) The Microarchitecture appear below the Architecture (ISA) ?
   Keep your answer brief (few lines) and to the point.

Ans:

a) The OS layer takes care of memory allocation, virtualisation of memory, context switching(call these set B) etc.. for a program in HLL. So for the OS to take care of such things, they(instruction format that CPU can understand, types of memory access etc.. , call this set A) need to be available on the before hand for it to implement things in set B. Architecture layer takes care of this set A. Architecture layer provides a defined way of crude memory access and operations syntax which are then used by OS to build a higher level of abstraction(set B) to make the life of programmers a bit more easy and use the resources efficiently. So OS is above architecture layer.

b) The ISA is the way in which we plan to give instructions to the computer, the exact syntax, definitions of the instruction. And how the computer is going to understand that is the job of microarchitecture which essentially mean that the way we built the internal logic design is dependent on the instruction sets we intend to use.ie., microarchitecture is kind of the hardware implementation of ISA. The ISA gives the information about the underlyiing microarchitecture in the machine, the way that CPU understands details. Those details provided by ISA can then be used to write codes in assembly language. In this way, ISA gives an higher level abstraction to microarchitecture and so is above microarchitecture.

2. In the table below, you need to fill in the column labelled “Code Density” (expressed in bytes). Assume that each machine (except for the stack machine) has 16 registers, that the width of the opcode field is 6 bits and that a memory address is 32 bits. Assume, for simplicity, that the only available addressing mode available on all machines is the direct mode. You are only expected to enter values in the “Code Density”column - no explanation is required.

No.	Architectural Style	No. Operands in ALU instr	Max. No. Memory operands in ALU instr	# instructions to compute x = ab+cde	Code density	Example
1	Stack	0	0	10	32	-
2	Accumulator-based	1	1	8	32	Intel 8085
3	Register-Memory	2	1	?	34	X86, IBM 360
4	Register-Memory	3	1	7	36	-
5	Memory-Memory	3	3	4	51	VAX

6	Memory-Memory	2	2	?	53	VAX
7	Register-register	3	0	10	41	MIPS, ARM

1. Push A 6+32
Push B 6+32
Mult 6
Push C 6+32
Push D 6+32
Mult 6
Push E 6+32
Mult 6
Add 6
Pop X 6+32
total = 6*10+32*6 = 60+192 = 252 bits, so 32 bytes

2. Load A 6+32
Mult B 6+32
Move R2 R1 (R1(Accumulator)->R2) 6+4+4
Load C 6+32
Mult D 6+32
Mult E 6+32
Add R2 6+4
Store X 6+32
total = 252 bits, so 32 bytes

3.Load R1 A 6+4+32
Mult R1 B 6+4+32
Load R2 C 6+4+32
Mult R2 D 6+4+32
Mult R2 E 6+4+32
Add R1 R2 6+4+4
Store R1 X 6+4+32
total = 266 bits, so 34 bytes

4.Load R1 A 6+4+32
Mult R1 R1 B 6+4+4+32
Load R2 C 6+4+32
Mult R2 R2 D 6+4+4+32
Mult R2 R2 E 6+4+4+32
Add R1 R1 R2 6+4+4+4
Store R1 X 6+4+32
total = 282 bits, so 36 bytes

5.Mult X1 A B 6+32+32+32
Mult X2 C D 6+32+32+32

Mult X2 X2 E 6+32+32+32
Add X3 X1 X2 6+32+32+32
total = 408 bits, so 51 bytes

6.Move X1 A 6+32+32
Mult X1 B 6+32+32
Move X2 C 6+32+32
Mult X2 D 6+32+32
Mult X2 E 6+32+32
Add X1 X2 6+32+32
total = 420 bits, so 53 bytes (X1 is the target memory where we want to store the result)

7. Load R1 A 6+4+32
Load R2 B 6+4+32
Mult R1 R1 R2 6+4+4+4
Load R2 C 6+4+32
Load R3 D 6+4+32
Mult R2 R2 R3 6+4+4+4
Load R3 E 6+4+32
Mult R2 R2 R3 6+4+4+4
Add R1 R1 R2 6+4+4+4
Store R1 X 6+4+32
total = 324 bits, so 41 bytes

3. In the program for the register-register architecture, we used 8 general purpose registers (GPRs). Rewrite the program assuming we had only 4 available GPRs. How many instructions would the program have?

Ans: 10

LD R1 A
LD R2 B
MULT R1 R1 R2
LD R2 C
LD R3 D
MULT R2 R2 R3
LD R3 E
MULT R2 R2 R3
ADD R1 R1 R2
ST R1 X