Welcome to the Class

Department of Computing and Information System

Programming Consideration

- Data processing operations normally requires up to 3 operands. For example, Z:= X + Y.
- The accumulator based CPU supports only single-address instructions.
- A program that implement Z:=X+Y, where X, Y and Z all refer to data words in M, can take the following form:

Programming Consideration

HDL format	Assembly language format	Comment
AC:=M(X)	LD X	Load X from M into AC
DR:=AC	MOV DR, AC	Move contents of AC to DR
AC:=M(Y)	LD Y	Load Y into AC
AC:=AC+DR	ADD	Add DR to AC
M(Z):=AC	ST Z	Store contents of AC in M

HDL = Hardware Description Language

VHDL = VHSIC HDL (VHSIC = Very High-Speed Integrated Circuits)

Programming Consideration

- Consider an instruction of form: AC:= f_i (AC, M (adr)) which require to move M(adr) to or from DR and one to perform the operation f_i.
- Memory reference complicates the instruction-decoding logic.
- Overall execution time should be reduced.

Programming Consideration

HDL format	Assembly language format	Comment
AC:=M(X)	LD X	Load X from M to AC
AC:=AC+M(Y)	ADD Y	Load Y into DR and add to AC
M(Z):=AC	ST Z	Store contents of AC in M

Instruction Set

The instruction set of the accumulator based CPU:

Type	Instruction	HDL format	Assembly format
Data Transfer	Load	AC:=M(X)	LD X
	Store	M(X):= AC	ST X
	Move Register	DR:=AC	MOV DR, AC
Data Processing	Add	AC:=AC+DR	ADD
	Subtract	AC:=AC-DR	SUB
	And	AC:=AC and DR	AND
	Not	AC:= not AC	NOT
Program Control	Branch	PC:=M (adr)	BRA adr
	Branch zero	if AC = 0 then PC:=M (adr)	BZ adr

Instruction Set

The arithmetic operation negation:

HDL format	Assembly language format	Comment
DR:=AC	MOV DR, AC	Copy contents X of AC to DR
AC:=AC-DR	SUB	Compute AC = X – X = 0
AC:=AC-DR	SUB	Compute AC = 0 – X = -X

A Multiplication Program

- Consider The program should implement AC:=AC×N where, the multiplicand is the initial content of AC and the multiplier N is a variable stored in memory.
- AC×N is implemented by executing the ADD instructions N times in the form AC+AC+……..+AC
- The memory location storing N acts as a count register and after each add operation, it is decremented by 1 until it reaches 0.
- The test for N=0 is performed by means of BZ instruction.
- The memory locations:

  one – stores constant 1

  mult– store N

  ac – store Y

  prod– partial product

A Multiplication Program

Line	Location	Instruction/ Data	Assembly format
0	one	00……..01	
1	mult	N	
2	ac	00……..00	
3	prod	00……..00	
4		ST ac	
5	Loop	LD mult	AC:= M (mult)
6		BZ exit	If AC= 0 then exit
7		LD one	AC:= M (one)
8		MOV DR, AC	DR:=AC
9		LD mult	AC:= M (mult)
10		SUB	AC:=AC-DR
11		ST mult	M (mult):= AC
12		LD ac	AC:= M (ac)
13		MOV DR, AC	DR:=AC
14		LD prod	AC:= M (prod)
15		ADD	AC:=AC+DR
16		ST prod	M (prod): =AC
17		BRA Loop	
18	exit		

Several Limitations of Accumulator-based CPU

- Because of few data registers in CPU, a considerable amount of time is spent shuttling the same information back and forth between the CPU and memory.
- It would both shorten the program and speed up its execution if it is possible to store the quantities 1, N, Y and P in their own CPU registers, as they are repeatedly required by the CPU.

Performance of a Computer

- **Response Time**

  - The time between the start and completion of a task. It is also known as **execution time**.
  - This includes disk accesses, memory accesses, I/O activities, operating system overhead, CPU execution time, etc.

- **Throughput**

  - The total amount of job done in a fixed amount of time.
  - Decreasing response time almost always improves throughput.

Relation between Performance and Execution Time

- $\text{Performance}_X = 1/\text{Execution time}_X$
- For two computers X and Y, if the performance of X is greater than the performance of Y, then

  $\text{Performance}_X > \text{Performance}_Y$

  $= 1/\text{Execution time}_X > 1/\text{Execution time}_Y$

  $= \text{Execution time}_Y > \text{Execution time}_X$
- If X is n times faster than Y, then

  $\text{Performance}_X / \text{Performance}_Y = n$

  or, $\text{Execution time}_Y / \text{Execution time}_X = n$

Relative Performance

- If computer A runs a program in 10 seconds and computer B runs the same program in 15 seconds, how much faster is A than B?

$\text{Performance}_A / \text{Performance}_B$

$= \text{Execution time}_B / \text{Execution time}_A$

$= 15/10$

$= 1.5$

A is 1.5 times faster than B.

Measuring Performance

- Time is the measure of computer performance.
- Computers are often shared. In such cases, the system may try to optimize throughput rather than minimizing the elapsed time for one program.
- **CPU execution time** or **CPU time** is the time the CPU spends computing a task not including the time spent waiting for I/O or running other programs.

Measuring Performance

- CPU time- 1. User CPU time
  2. System CPU time

- CPU time spent in the program is called User CPU time and the CPU time spent in the operating system performing tasks requested by the program is called System CPU time.

Measuring Performance

- Suppose, after executing a program we get the following measurements:

  User CPU time – 90.7 seconds

  System CPU time – 12.9 seconds

  Elapsed time – 159 seconds

  percentage of elapsed time is (90.7+12.9)/159 = 65%

- The term **system performance** is used to refer to elapsed time on an unloaded system while **CPU performance** refers to user CPU time on an unloaded system.

Clock Cycles

- Almost all computers are constructed using a clock that runs at a constant rate and determines when events take place in the hardware.
- Clock period is the time for a complete clock cycle.
- Clock rate is the inverse of clock period.
- CPU time for a program can be expressed in two ways:

  CPU time = CPU clock cycles for a program X clock cycle time

  o CPU time = CPU clock cycle for a program / Clock rate

CPU Time

- CPU time for a program can be expressed in two ways:

  CPU time = CPU clock cycles for a program X clock cycle time

  or

  CPU time = CPU clock cycle for a program / Clock rate

- If we know the number of clock cycles and the number of instructions execute (Instruction count, IC), we can calculate the average number of clock cycles per instruction (CPI).

  CPI = CPU clock cycles for a program / Instruction count

- Instructions per clock (IPC) is the inverse of CPI.

CPU Time

- **CPU time = Instruction count X Cycles per instruction X Clock cycle time**

or

CPU time = Instruction count X Cycles per instruction / Clock rate

$T = N \times CPI / f$

- It is possible to compute the CPU clock cycles by looking at the different types of instructions and using their individual clock cycle counts.

$$\text{CPU clock cycles} = \Sigma(CPI_i \times C_i)$$

where C_i is the count of the number of instructions of class i executed, CPI_i is the average number of cycles per instruction for that instruction class, and n is the number of instruction classes.

Comparing Code Segments

	CPI for this instruction class		
	A	B	C
CPI	I	2	3

Code Sequence	Instruction count for instruction class		
	A	B	C
1	2	1	2
2	4	1	1

a) Which code sequence executes the most instructions?

b) Which will be faster?

c) What is the CPI for each sequence?

a) Sequence 1 executes 5 instructions and sequence 2 executes 6 instructions.

b) CPU clock cycles$_1$ = (2X1)+(1X2)+(2X3) = 10 cycles

CPU clock cycles$_2$ = (4X1)+(1X2)+(1X3) = 9 cycles

So code sequence 2 is faster.

c) Since sequence 2 takes fewer overall clock cycles but has more instruction, it must have a lower CPI.

$CPI_1 = 10/5 = 2$

$CPI_2 = 9/6 = 1.5$

CPU Time

- CPU performance depends on three characteristics:
  - **Clock cycle time (Clock rate) :** Hardware technology
  - **Clock cycles per instruction:** Architecture
  - **Instruction count (N) :** software technology

Components affect the CPU Performance

Hardware or software component	Affects what?
Algorithm	Instruction count
Programming language	Instruction count CPI
Compiler	Instruction count CPI
Instruction set architecture	Instruction count Clock rate CPI

Speedup Techniques

Feature	Objective
Cache Memory	To provide the CPU with faster access to instruction and data.
Pipelined Processing	To increase performance by allowing the processing of several instructions to be partially overlapped.
Superscalar Processing	To increase performance by allowing several instructions to be processed in parallel (full overlapping).

Any Question???