

The Microprogram in Full

		Microinstruction																						
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
Micromemory address	0																	1				1		1
	1																					1		1
	2								1		1											1		
	3			1				1						1						1				1
	4	1				1		1		1										1				
	5																	1		1	1			1

C.Sc. 131: Systems Architecture - 2006

Phase 5

C.Sc. 131: Systems Architecture - 2006

Evaluation of Microinstructions

- Writing programs in microcode is very difficult:
 - cf code required to multiply two numbers
 - each instruction performs only a very small task.
 - instructions are closely tied to machine architecture.
 - instructions must be coded in binary.

C.Sc. 131: Systems Architecture - 2006

Providing More Support

- If we are going to write big programs we need to provide programmers with easier to use and more powerful instructions.
- Simplest languages which offer such support are called machine languages.
 - still at a very low level but much easier to use than microcode.
 - they introduce limited abstractions over the machine's hardware, e.g. the accumulator.

C.Sc. 131: Systems Architecture - 2006

Overview of a Simple Machine Language

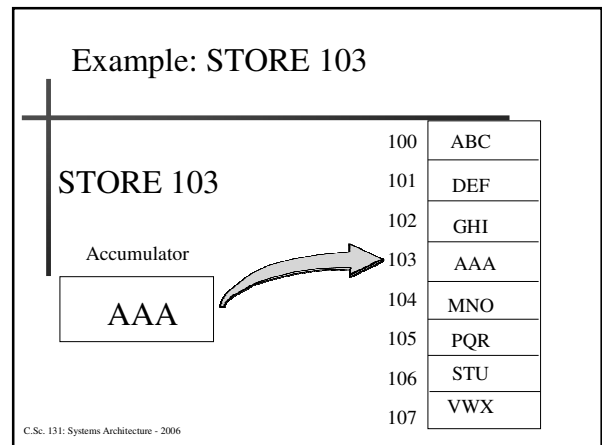
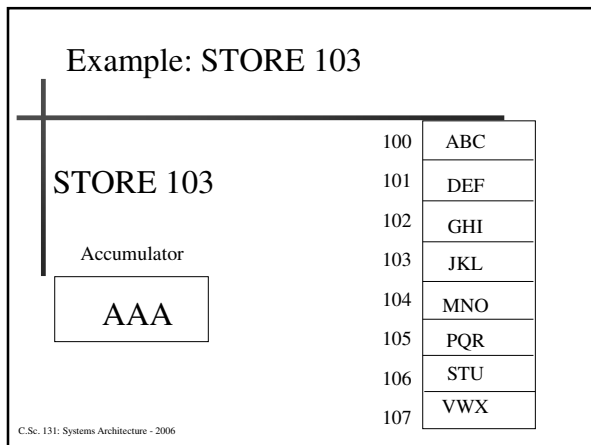
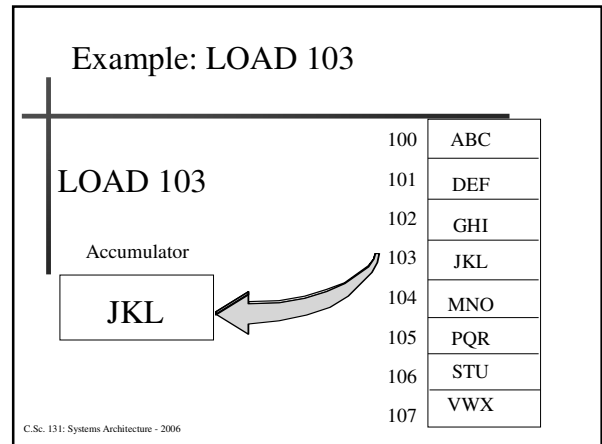
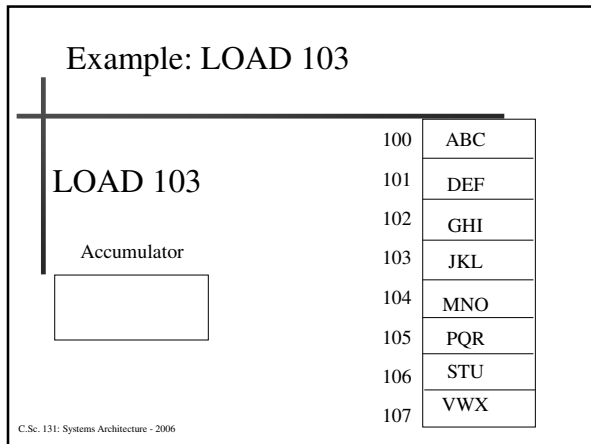
- Simple language involving only 11 instructions.
- Main abstraction is the accumulator which stores operands and results of arithmetic operations.
- Why? - Because arithmetic operations often use the result of a preceding operation as an operand.

C.Sc. 131: Systems Architecture - 2006

Operations 1 & 2

- Load
 - moves values from memory to the accumulator.
 - LOAD 123
- Store
 - moves values from the accumulator into memory.
 - STORE 123

C.Sc. 131: Systems Architecture - 2006



- ### Operations 3,4,5 & 6
- Add
 - Adds the value from memory to the accumulator.
 - ADD 123
 - Subtract
 - Subtracts the value from memory from the accumulator.
 - SUBTRACT 123
 - Multiply
 - MULTIPLY 123
 - Divide
 - DIVIDE 123
- C.Sc. 131: Systems Architecture - 2006

The Machine Language So Far

Operation	Explanation
LOAD	M -> ACC
STORE	ACC -> M
ADD	ACC + M -> ACC
SUBTRACT	ACC - M -> ACC
MULTIPLY	ACC * M -> ACC
DIVIDE	ACC / M -> ACC

C.Sc. 131: Systems Architecture - 2006

Control Transfer (Ops 7,8 & 9)

- Unconditional transfer of control.
 - JUMP M : Start executing instruction at memory cell M.
- Conditional transfer of control
 - JUMPZERO M : Start executing instruction at memory cell M if the accumulator contains zero.
 - JUMPMSB M : Start executing instruction at memory cell M if the msb of the accumulator = 1.

C.Sc. 131: Systems Architecture - 2006

The Machine Language So Far

Operation	Explanation
LOAD	M -> ACC
STORE	ACC -> M
ADD	ACC + M -> ACC
SUBTRACT	ACC - M -> ACC
MULTIPLY	ACC * M -> ACC
DIVIDE	ACC / M -> ACC
JUMP	Jump to cell M
JUMPZERO	Jump if ACC = 0
JUMPMSB	Jump if msb = 1

C.Sc. 131: Systems Architecture - 2006

Op Codes and Operands

- We need to identify each instruction by giving it an op-code.
- We use 4 bits because we have 11 instructions.
- We also need to store the address of the operand.

0001
0000 0000 0001
 Op-code Operand

C.Sc. 131: Systems Architecture - 2006

The Machine Language So Far

opcode	adsss	Operation	Explanation
0001		LOAD	M -> ACC
0010		STORE	ACC -> M
0011		ADD	ACC + M -> ACC
0100		SUBTRACT	ACC - M -> ACC
0101		MULTIPLY	ACC * M -> ACC
0110		DIVIDE	ACC / M -> ACC
0111		JUMP	Jump to cell M
1000		JUMPZERO	Jump if ACC = 0
1001		JUMPMSB	Jump if msb = 1

C.Sc. 131: Systems Architecture - 2006

An Example

Address	Instruction	Op Code	Operand Addr
1	1005	1	5
2	5005	5	5
3	2005	2	5
4	FFFF		
5	000A		

- Calculates $x*x$ where x is initially in location 5.
- Compare this with its microcode program.

C.Sc. 131: Systems Architecture - 2006

An Example

ACC : 0

Address	Instruction
1	1005
2	5005
3	2005
4	FFFF
5	000A

- Calculates $x*x$ where x is initially in location 5.
- Compare this with its microcode program.

C.Sc. 131: Systems Architecture - 2006

An Example ACC : A_{16}

Address	Instruction
1	1005 LOAD 5 -> ACC
2	5005
3	2005
4	FFFF
5	000A

- Calculates $x*x$ where x is initially in location 5.
- Compare this with its microcode program.

C.Sc. 131: Systems Architecture - 2006

An Example ACC : 64_{16}

Address	Instruction
1	1005 Contents 5 -> ACC
2	5005 ACC * Contents 5 -> ACC
3	2005
4	FFFF
5	000A

- Calculates $x*x$ where x is initially in location 5.
- Compare this with its microcode program.

C.Sc. 131: Systems Architecture - 2006

An Example ACC : 64_{16}

Address	Instruction
1	1005 Contents 5 -> ACC
2	5005 ACC * Contents 5 -> ACC
3	2005 Contents of Acc -> 5
4	FFFF
5	0064

- Calculates $x*x$ where x is initially in location 5.
- Compare this with its microcode program.

C.Sc. 131: Systems Architecture - 2006

An Example ACC : 64_{16}

Address	Instruction
1	1005 Contents 5 -> ACC
2	5005 ACC * Contents 5 -> ACC
3	2005 Contents of Acc -> 5
4	FFFF Stop
5	0064

- Calculates $x*x$ where x is initially in location 5.
- Compare this with its microcode program.

C.Sc. 131: Systems Architecture - 2006

The Machine Language So Far

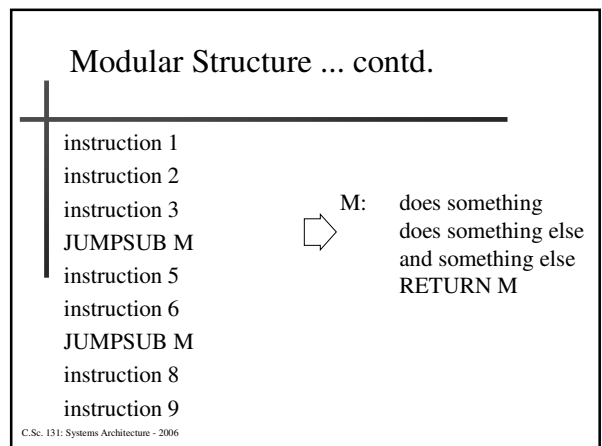
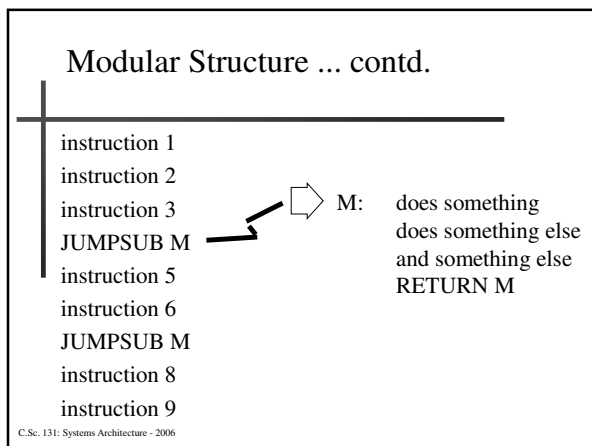
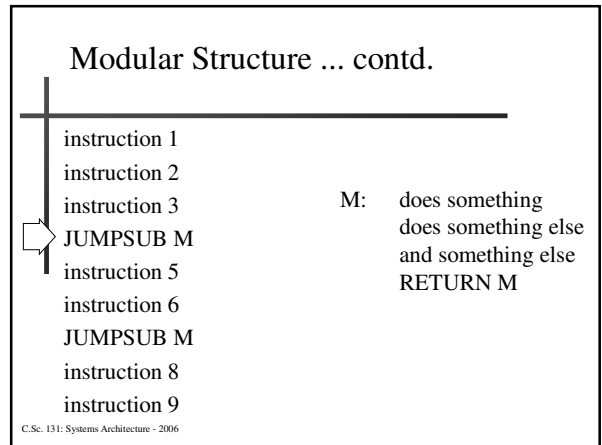
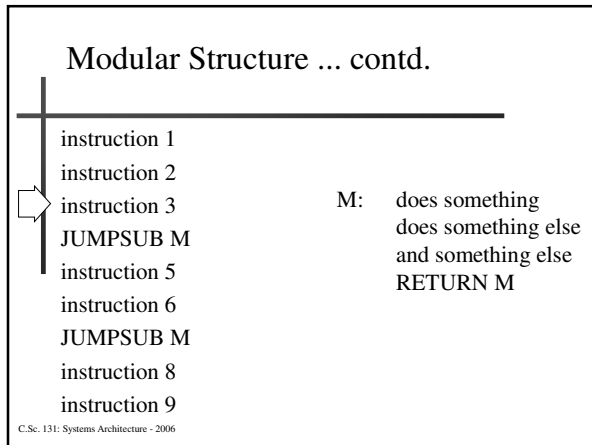
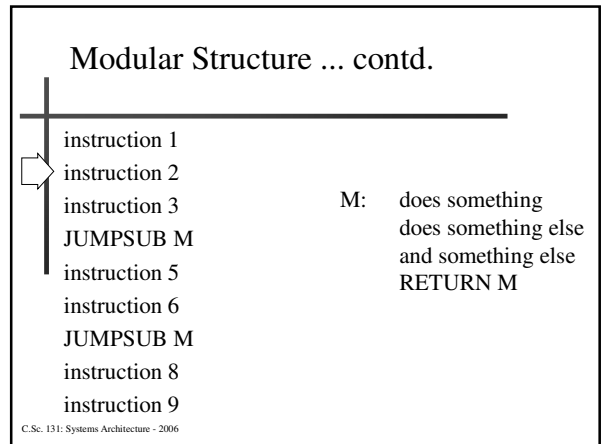
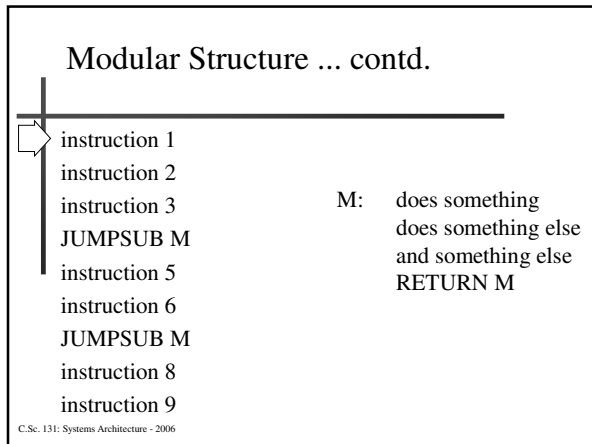
opcode	adss	Operation	Explanation
0001		LOAD	M -> ACC
0010		STORE	ACC -> M
0011		ADD	ACC + M -> ACC
0100		SUBTRACT	ACC - M -> ACC
0101		MULTIPLY	ACC * M -> ACC
0110		DIVIDE	ACC / M -> ACC
0111		JUMP	Jump to cell M
1000		JUMPZERO	Jump if ACC = 0
1001		JUMPM5B	Jump if msb = 1

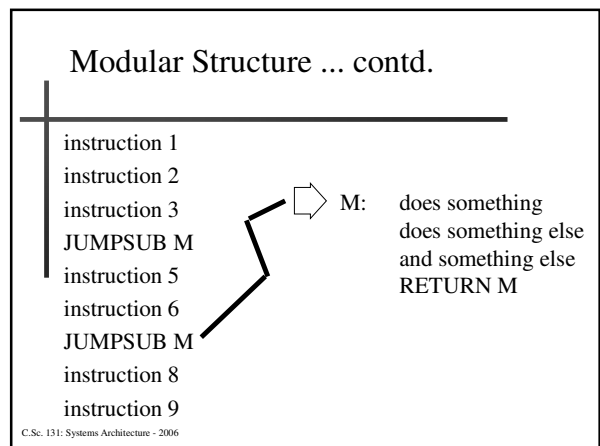
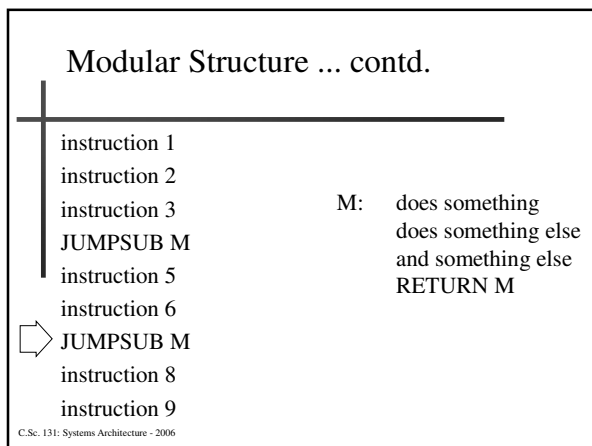
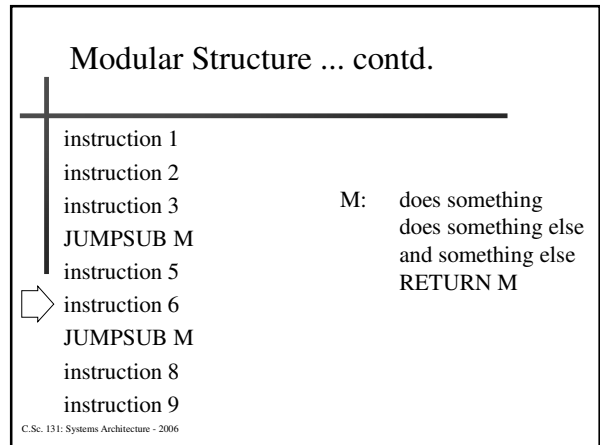
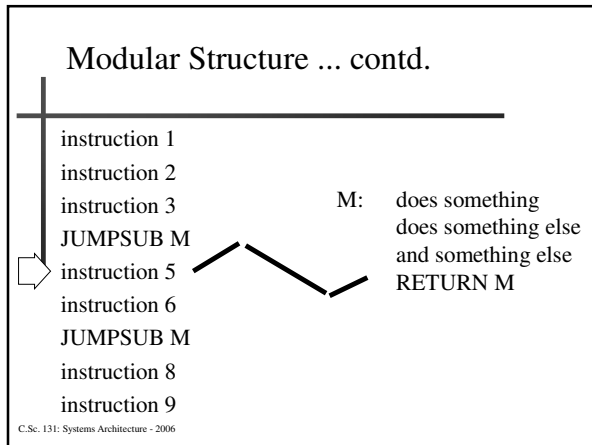
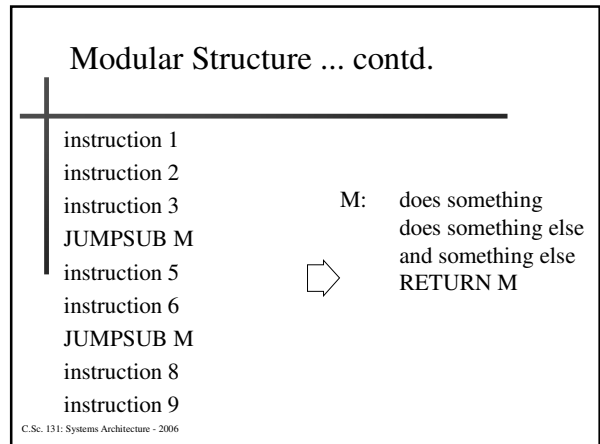
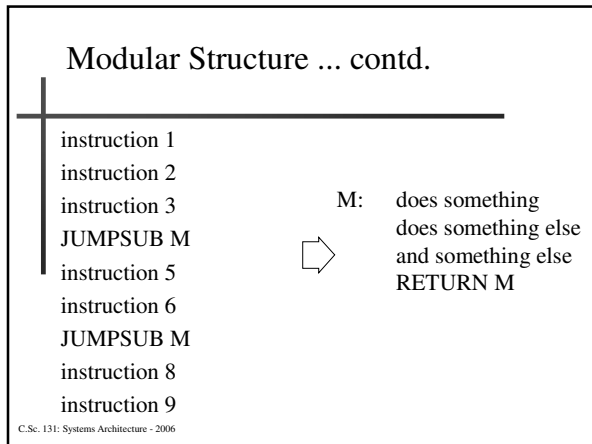
C.Sc. 131: Systems Architecture - 2006

Modular Structure (Operations 10 & 11)

- Two instructions not yet described, i.e. JUMPSUB and RETURN
- These instructions may be used to structure programs, i.e. subroutines.
- Equivalent to gosubs (BASIC) or function calls.

C.Sc. 131: Systems Architecture - 2006





Modular Structure ... contd.

instruction 1
 instruction 2
 instruction 3
 JUMPSUB M
 instruction 5
 instruction 6
 JUMPSUB M
 instruction 8
 instruction 9

⇒ M: does something
 does something else
 and something else
 RETURN M

C.Sc. 131: Systems Architecture - 2006

Modular Structure ... contd.

instruction 1
 instruction 2
 instruction 3
 JUMPSUB M
 instruction 5
 instruction 6
 JUMPSUB M
 instruction 8
 instruction 9

⇒ M: does something
 does something else
 and something else
 RETURN M

C.Sc. 131: Systems Architecture - 2006

Modular Structure ... contd.

instruction 1
 instruction 2
 instruction 3
 JUMPSUB M
 instruction 5
 instruction 6
 JUMPSUB M
 instruction 8
 instruction 9

⇒ M: does something
 does something else
 and something else
 RETURN M

C.Sc. 131: Systems Architecture - 2006

Modular Structure ... contd.

instruction 1
 instruction 2
 instruction 3
 JUMPSUB M
 instruction 5
 instruction 6
 JUMPSUB M
 instruction 8
 instruction 9

⇒ M: does something
 does something else
 and something else
 RETURN M

C.Sc. 131: Systems Architecture - 2006

Modular Structure ... contd.

instruction 1
 instruction 2
 instruction 3
 JUMPSUB M
 instruction 5
 instruction 6
 JUMPSUB M
 instruction 8
 instruction 9

⇒ M: does something
 does something else
 and something else
 RETURN M

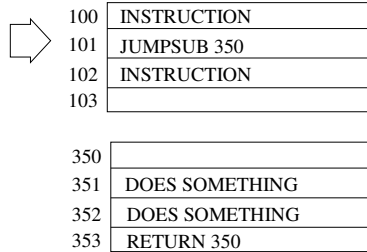
C.Sc. 131: Systems Architecture - 2006

Implementing JUMPSUB & RETURN

100	INSTRUCTION
101	JUMPSUB 350
102	INSTRUCTION
103	
350	
351	DOES SOMETHING
352	DOES SOMETHING
353	RETURN 350

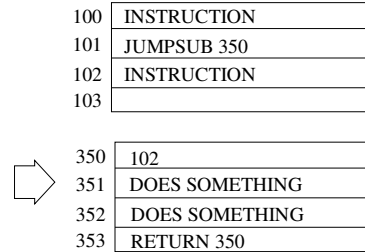
C.Sc. 131: Systems Architecture - 2006

Implementing JUMPSUB & RETURN



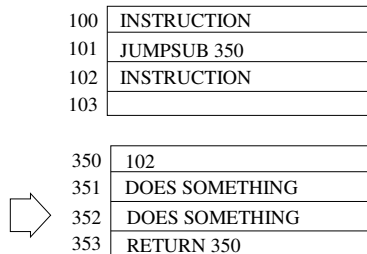
C.Sc. 131: Systems Architecture - 2006

Implementing JUMPSUB & RETURN



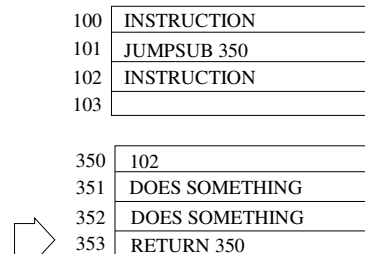
C.Sc. 131: Systems Architecture - 2006

Implementing JUMPSUB & RETURN



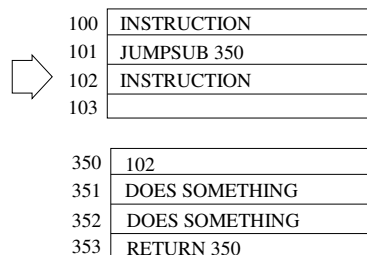
C.Sc. 131: Systems Architecture - 2006

Implementing JUMPSUB & RETURN



C.Sc. 131: Systems Architecture - 2006

Implementing JUMPSUB & RETURN



C.Sc. 131: Systems Architecture - 2006

Supporting Machine Language Instructions

- Early machines were built to execute machine language instructions directly:
 - this is expensive in design terms.
 - lots of duplication since many instructions have common functionality.
- By contrast, microinstructions are much simpler and machines may be built to execute microinstructions quite cheaply.

C.Sc. 131: Systems Architecture - 2006

Supporting Machine Language Instructions

- Solution ?
- Provide a means of converting from machine language instructions to microcode.
- This can be done in software; the program is called a microprogrammed interpreter.

C.Sc. 131: Systems Architecture - 2006

A Microprogrammed Interpreter

- A microprogrammed interpreter is supplied with every computer.
- The interpreter is fixed in the machine's micromemory and maps machine language instructions in main memory into microinstructions.
- The interpreter is normally the only algorithm ever microprogrammed - programmers normally only ever see machine language.

C.Sc. 131: Systems Architecture - 2006

Basic Algorithm for a Microprogrammed Interpreter

```
procedure interpreter is
begin
  while (true) do
    fetch next machine language from address in B
    add 1 to contents of B
    decode the instruction
    execute the instruction
  end while
end
```

C.Sc. 131: Systems Architecture - 2006

Notes on the Interpreter

- The interpreter is performing what is called a fetch-execute cycle.
- The instruction which the algorithm fetches is contained within the memory location whose address is held in register B: the program counter,
- Jumps to new instructions are caused by changing the contents of register B.

C.Sc. 131: Systems Architecture - 2006

Fetching Continued...

100	100D
101	200C
102	100B
103	800E

- If register B holds the value 101
 - 200C is the instruction fetched
- If increment register B to hold the value 102
 - 100B is the instruction fetched

C.Sc. 131: Systems Architecture - 2006

Basic Algorithm for a Microprogrammed Interpreter

```
procedure interpreter is
begin
  while (true) do
    fetch next machine language from address in B
    add 1 to contents of B
    decode the instruction
    execute the instruction
  end while
end
```

C.Sc. 131: Systems Architecture - 2006

Summary

- Microprogramming is too low level - it takes a long time to write even a simple program.
- Machine languages are better, so we build microprogram driven computers which can interpret machine language programs.
- The interpreter performs what is commonly called the fetch-execute cycle.

C.Sc. 131: Systems Architecture - 2006

Next Lecture...

- More machine code and a new language: assembly language.
- D & L pp 215- 222 but note that these pages are very much background reading.

C.Sc. 131: Systems Architecture - 2006