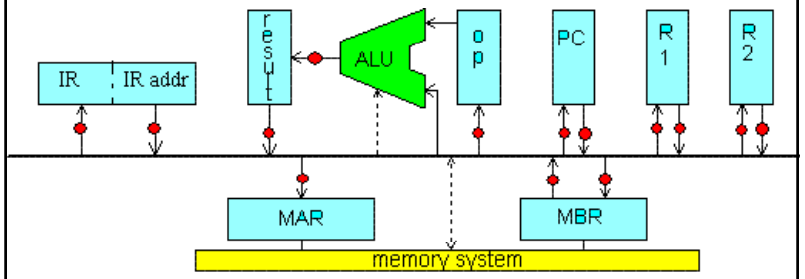
Microcode

COMP375 Computer Architecture and Organization

Goals

- Understand how microcode directs register to bus connection switches
- Understand how the CPU control unit directs the execution of simple instructions

Simple CPU



Parts of the Simple CPU

- Program Counter
- User registers, R1 and R2
- Instruction register, including the address field
- ALU – input from bus and operand register
  - Operand register – holds second operand
  - Result register – saves ALU output

Memory Interface

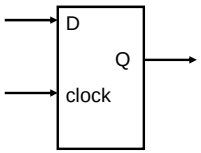
- Memory Address Register (MAR)
- Memory Buffer Register (MBR)
- To read data from memory, the CPU puts the address in the MAR and signals the memory system to read. Data from memory is loaded in the MBR.
- To write data to memory, the address is put in the MAR and the data in the MBR. The memory system is set to write.
- Memory access is slower than the CPU.

The time to access memory relative to the time to move data between registers is:

1. Registers are faster
2. Memory is faster
3. With cache the difference is insignificant
4. All of the above
5. None of the above
6. Both answers 4 & 5

Register Organization

- A register is a set of flip-flops. There is one flip-flop for each bit.
- The output of a D flip-flop is the same as the input the last time the clock was true.

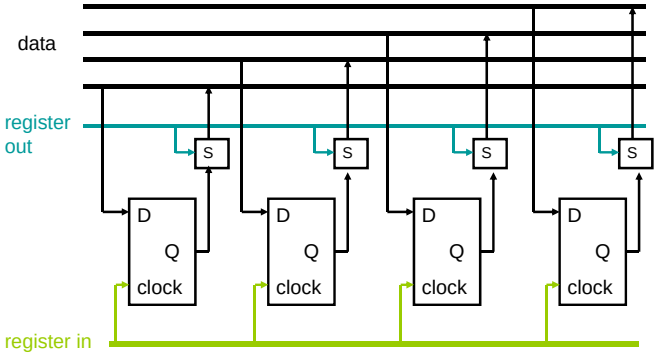


If the clock is true, Q = D

if the clock is false, Q = last value of D

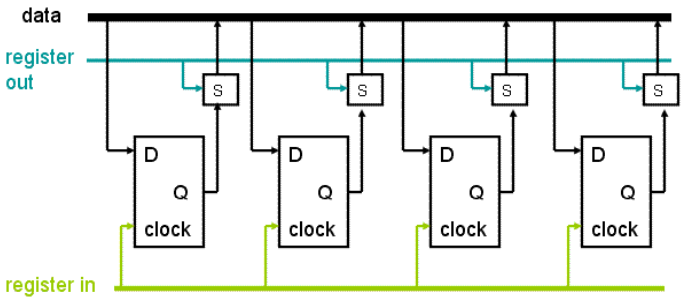
Register Detail

- A register created from four flip-flops



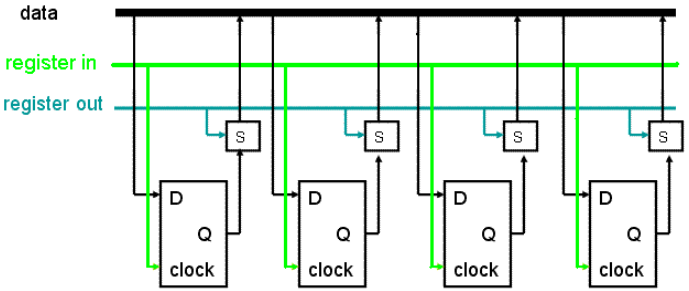
Register Detail

- Drawing all bus wires as one line



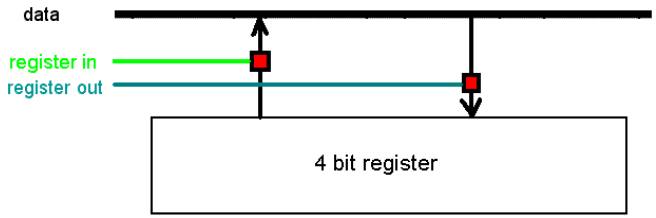
Register Detail

- Move the “register in” wire to the top



Register Detail

- Draw the four flip-flops as one register



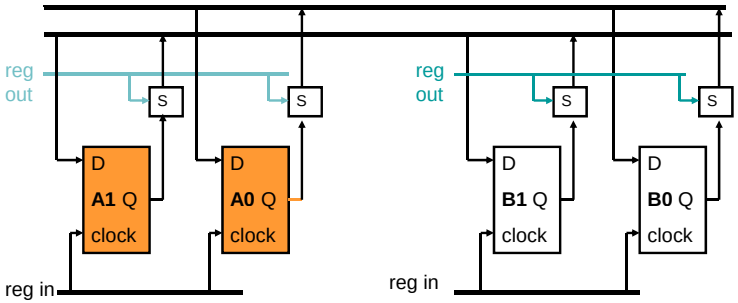
Register Copy

To move a value from register A to register B *(which can be any user or system register)*.

- Turn on the “register out” switch for register A
- The output of the register A flip-flops sets the value on the bus.
- Turn on the “register in” switch to set the clock input of the register B flip-flops to True.
- The values on the bus (from register A) connect to the input of register B’s flip-flops.
- Turn off “register in” and “register out”.

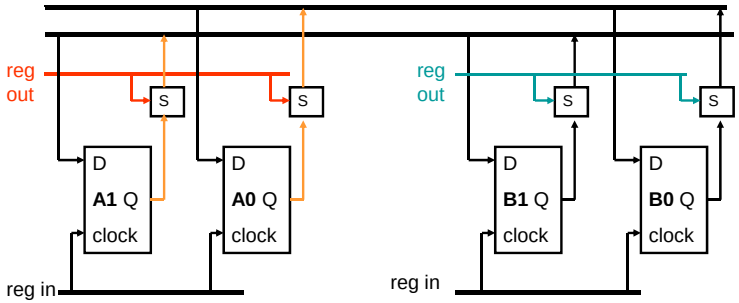
2 Bit Register Copy

- The goal is to move the two bit value in register A to register B



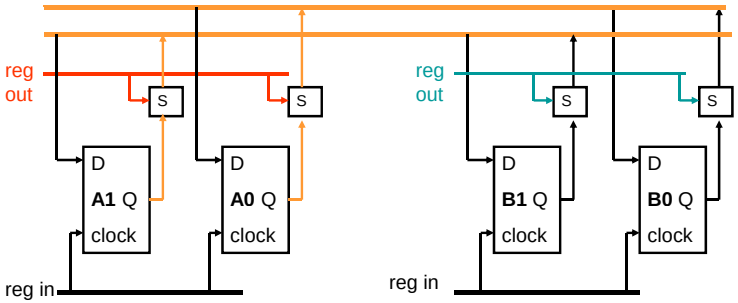
2 Bit Register Copy

- Turn on the “register out” switch for register A



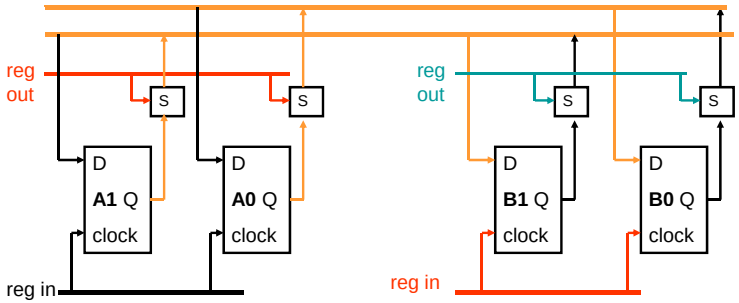
2 Bit Register Copy

- The output of the register A flip-flops sets the value on the bus.



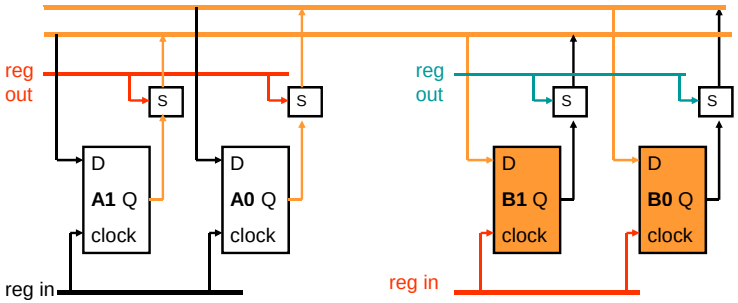
2 Bit Register Copy

- Turn on the “register in” switch to set the clock input of the register B flip-flops to True.



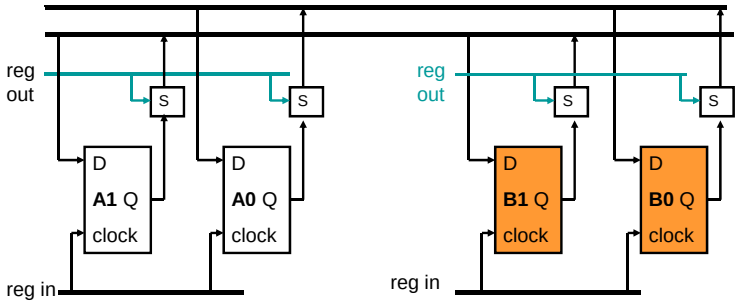
2 Bit Register Copy

- The values on the bus (from register A) connect to the input of register B's flip-flops.



2 Bit Register Copy

- Turn off "register in" and "register out".



Processor Operation

- The opcode of the instruction defines what operation should be done.
- Most of the internal operation of the CPU is moving data and addresses between registers and other units.
- Moving data across the bus only requires turning on and off the proper switches in the proper sequence.

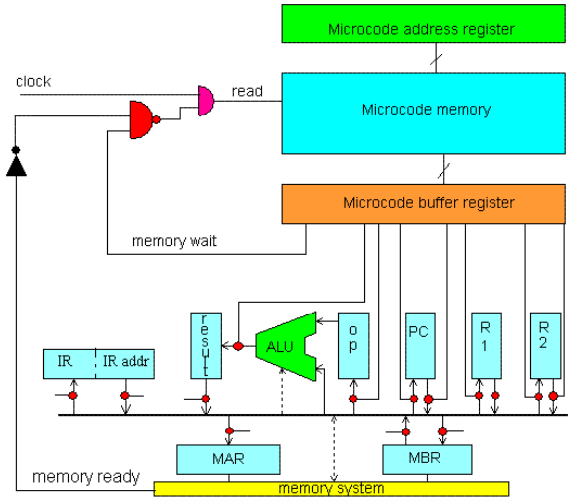
Control Words

- Each switch controlling the flow of data to and from the bus is connected by a control line.
- Sometimes a single control line goes to several switches.
- A control word defines the state of the control lines.
- The processor executes an instruction by sequencing through a set of control words.

Control Word to copy R2 to the Operand Register

bus → IR	IR adr → bus	result → bus	ALU → result	ALU fun	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	Mem func
					X						X				

Microcode Control



Representing Microcode

- In our class we will show the microcode as a table with rows of control lines.
- Each row is executed in time from top to bottom.
- The table represents the Microcode Store.

bus → IR	IR adr → bus	result → bus	ALU → result	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU fun	Mem func
						X					X				read
															wait
X													X		

Microcode in VHDL

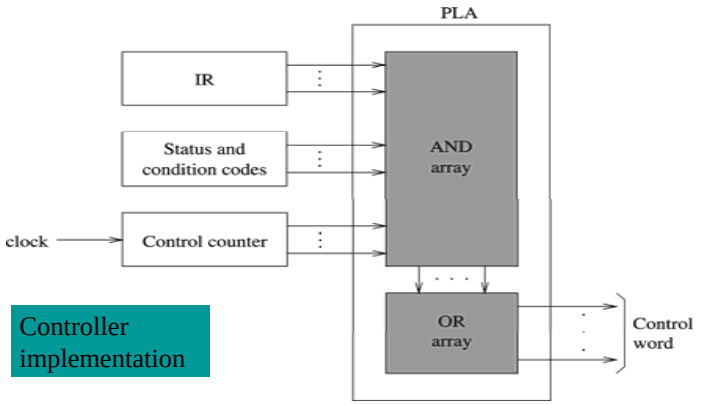
- Microcode is often described using VHSIC hardware description language (VHDL)

```
process(A,B,S)
  begin if S = '1' then
     X <= A;
  else
     X <= B;
  end if;
end process;
```

Processor Control

- Hardware Control
  - Logic gates within the CPU activate the appropriate switches in sequence.
  - Often used in RISC processors.
  - Hardware implementation is complex and expensive
- Software Control
  - A microcode program controls the sequence of control words.
  - Often used in CISC processors

Hardware Control

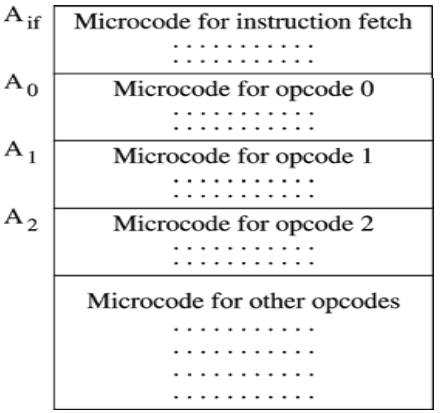


Microprogrammed Control

- A microcode program is a sequence of microinstruction steps that generate the control signals
  - Encode the signals of each step as a codeword
    - Called *microinstruction*
  - An instruction is expressed by a sequence of codewords

Microcode Store

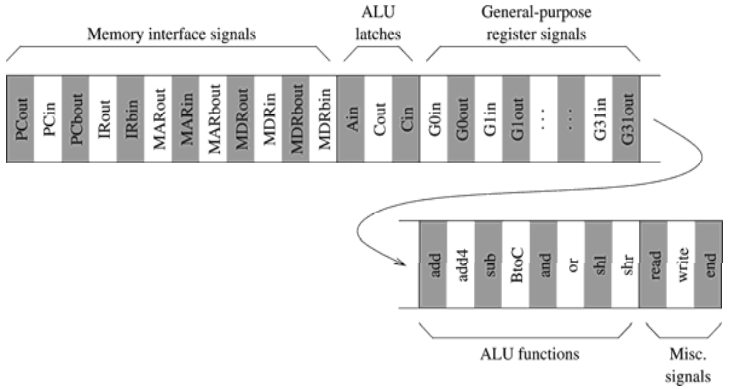
The microcode for the processor is kept in the control store. This is read-only memory in the CPU.



Microinstruction format

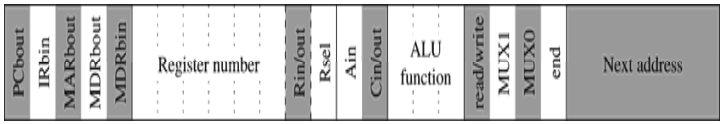
- Two basic ways
  - Horizontal organization
  - Vertical organization
- Horizontal organization
  - One bit for each signal
  - Very flexible
  - Long microinstructions

Horizontal Microcode Format



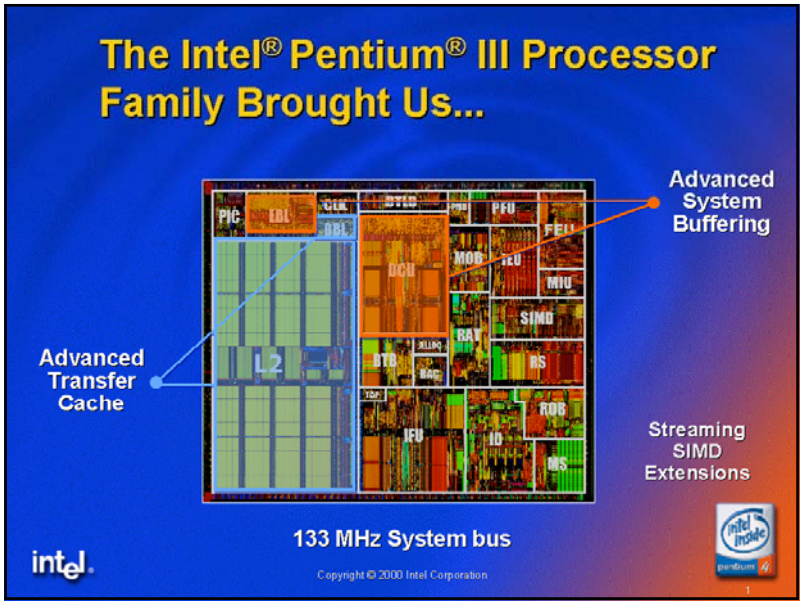
Vertical Organization

- Encodes to reduce microinstruction length
  - Reduced flexibility
- Example:
  - Horizontal organization
    - 64 control signals for the 32 general purpose registers
  - Vertical organization
    - 5 bits to identify the register and 1 for in/out

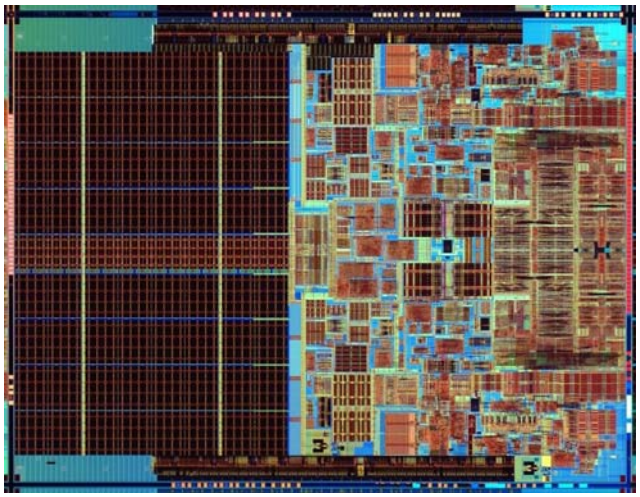


Microcode

- Each machine language instruction can be defined by a series of switch settings.
- The switch settings can be determined by combinatorial circuitry or by microcode.
- Microcode is a very low level program that sets the switches and settings to implement the machine language instructions.
- The microcode is stored in a special ROM on the CPU chip.

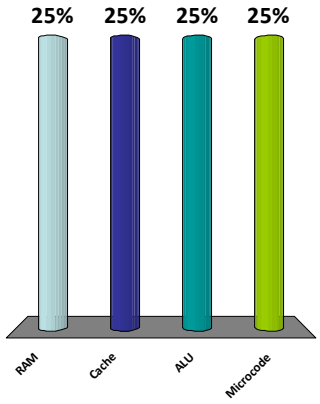


Intel Core 2 Extreme Processor



What is the function of the left side of that processor?

1. RAM
2. Cache
3. ALU
4. Microcode



Memory Read

- Copy the address to read into the Memory Address Register.
- Tell the memory system to “READ”
- A following microcode instruction must “WAIT” until the read is complete.
- Copy the data from the Memory Buffer Register to the desired register or ALU.

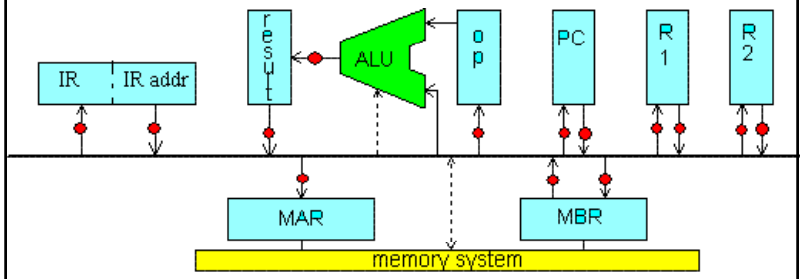
Memory Write

- Copy the address to be written into the Memory Address Register.
- Copy the data into the Memory Buffer Register.
- Tell the memory system to “WRITE”.
- A future microcode instruction must “WAIT” before doing another memory function.

Fetch Execute Cycle

1. Fetch the instruction from the memory address in the Program Counter register
2. Increment the Program Counter
3. Decode the type of instruction
4. Fetch the operands
5. Execute the instruction
6. Store the results

Simple CPU



Instruction Fetch

1. Fetch the instruction from the memory address in the Program Counter register
- Copy the program counter to the Memory Address Register
- Tell the memory system to read.
- Wait for the read to complete
- Copy from the Memory Data Register to the Instruction Register

Instruction Fetch

bus → IR	IR adr → bus	result → bus	ALU → result	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU fun	Mem func
						X					X				read
															wait
X													X		

2. Increment the Program Counter

- Copy the Program Counter to the ALU input register.
- Set the ALU function to increment.
- Copy the ALU output register to the Program Counter

2. Increment the Program Counter

bus → IR	IR adr → bus	result → bus	ALU → result	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU fun	Mem func
			X			X								inc	
		X			X										

4. Fetch the operands

Assuming direct addressing

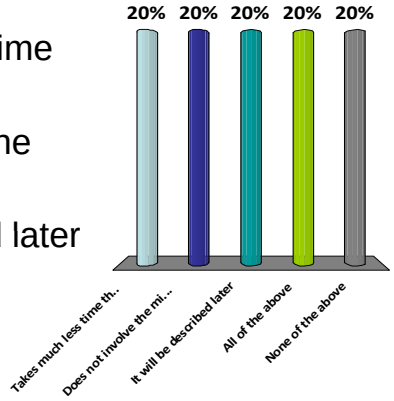
- Copy the address portion of the instruction register to the Memory Address Register.
- Tell the memory system to read.
- Wait for the read to complete
- Copy from the Memory Data Register to the appropriate data or ALU Register

4. Fetch the operands

bus → IR	IR adr → bus	result → bus	ALU → result	bus → opnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU func	Mem func
	X										X				read
															wait
							?						X		

What happened to step 3 decode operand?

1. Takes much less time than other steps
2. Does not involve the microcode
3. It will be described later
4. All of the above
5. None of the above



5. Execute the instruction

Assume an arithmetic instruction

- Copy the operand from a data register to an ALU input register.
- Set the ALU function according to the opcode field of the instruction register

5. Execute the instruction

bus → IR	IR adr → bus	result → bus	ALU → result	bus → opnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU func	Mem func
				X				X						add	
			X							X					

Assume adding R1 and R2

6. Store the results

- Copy the output ALU register to the appropriate data register.

6. Store the results

bus → IR	IR adr → bus	result → bus	ALU → result	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU func	Mem func
		X					X								

Add R1, xyz (memory direct)

bus → IR	IR adr → bus	result → bus	ALU → result	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU func	Mem func
						X					X				read
															wait
	X										X				
			X			X								inc	
		X			X										
	X										X				read
				X				X							wait
			X										X	add	
		X					X								

What control lines are set to copy the value from the result reg to R1?

bus → IR	IR adr → bus	result → bus	ALU → result	ALU fun	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	Mem func
		A			B			C		D	E				

1. A & E
2. B & C
3. A & D
4. C & A

Layers

- Applications
- Middleware
- High level languages
- Machine Language
- Microcode
- Logic circuits
- Gates
- Transistors
- Silicon structures

Example Layers

C++

A = A + B;

Assembler

Load R1, A
Add R1, B
Store R1, A

Load R1, A

Instruction fetch not shown

bus → IR	IR adr → bus	result → bus	ALU → result	ALU fun	bus → opnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	Mem func
	X											X			read
															wait
								X						X	

Add R1, B

Instruction fetch not shown

bus → IR	IR adr → bus	result → bus	ALU → result	ALU fun	bus → opnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	Mem func
	X											X			read
					X				X						wait
			X	add										X	
		X						X							

Store R1, A

Instruction fetch not shown

bus → IR	IR adr → bus	result → bus	ALU → result	ALU fun	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	Mem func
	X											X			
									X				X		write
															wait