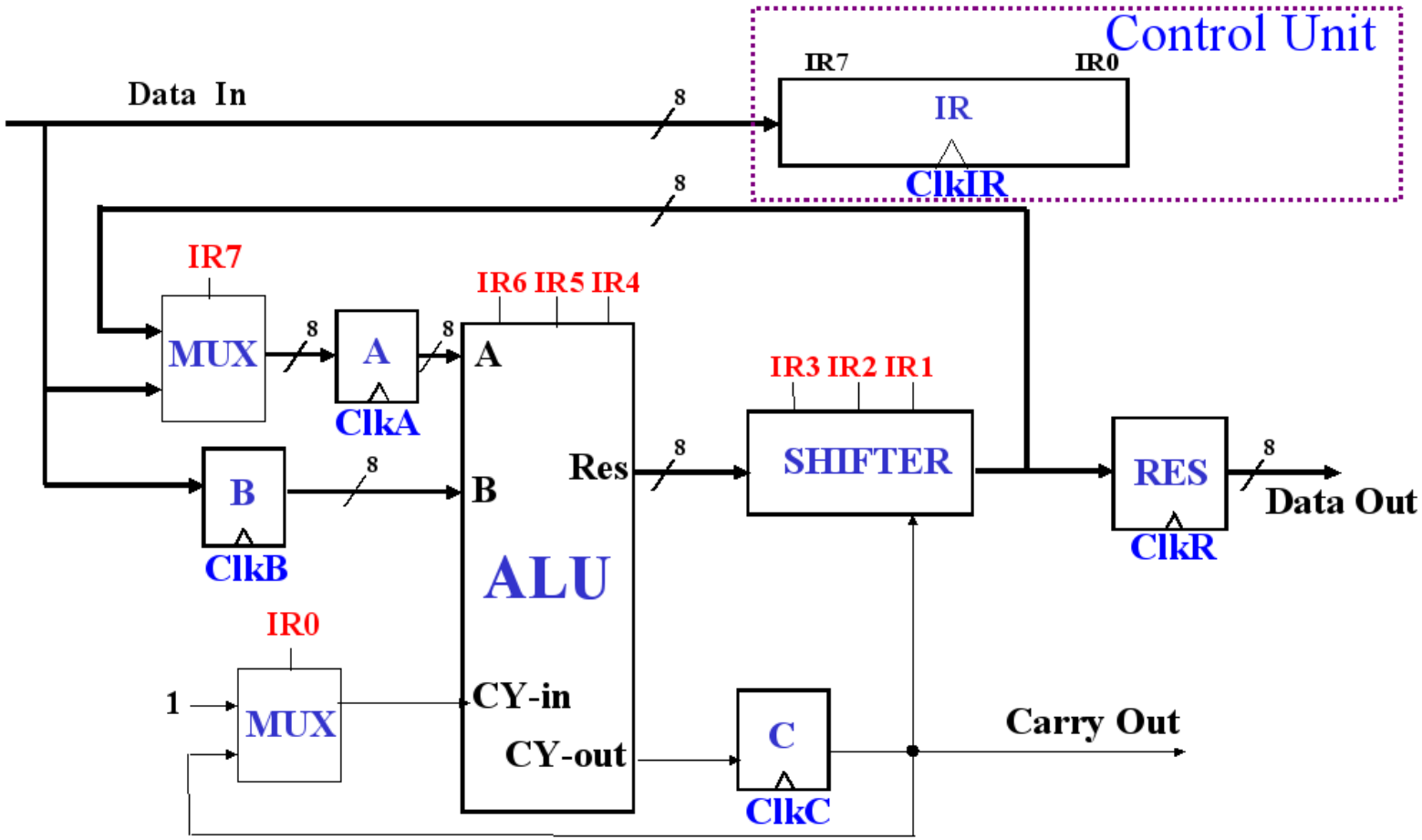
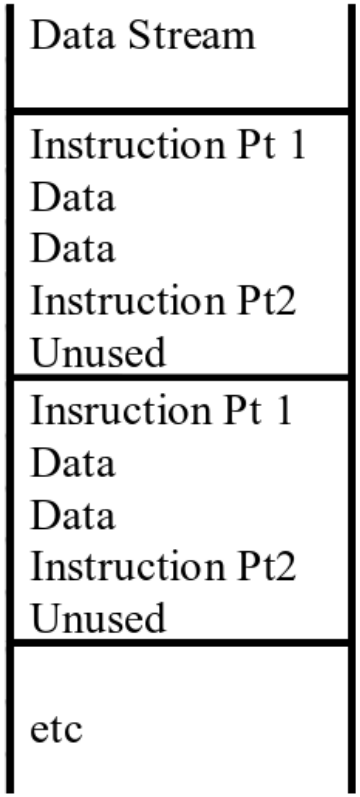
Lecture 17

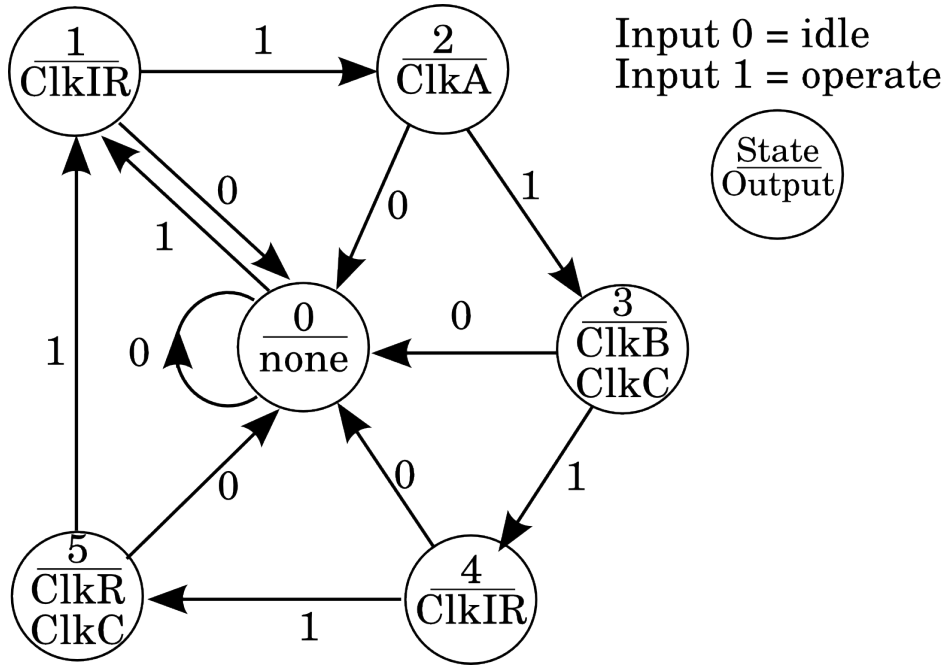
A 32-bit Processor - Architecture and Instructions

The Manual Processor - Data path Diagram



The manual Processor - Execution Cycle

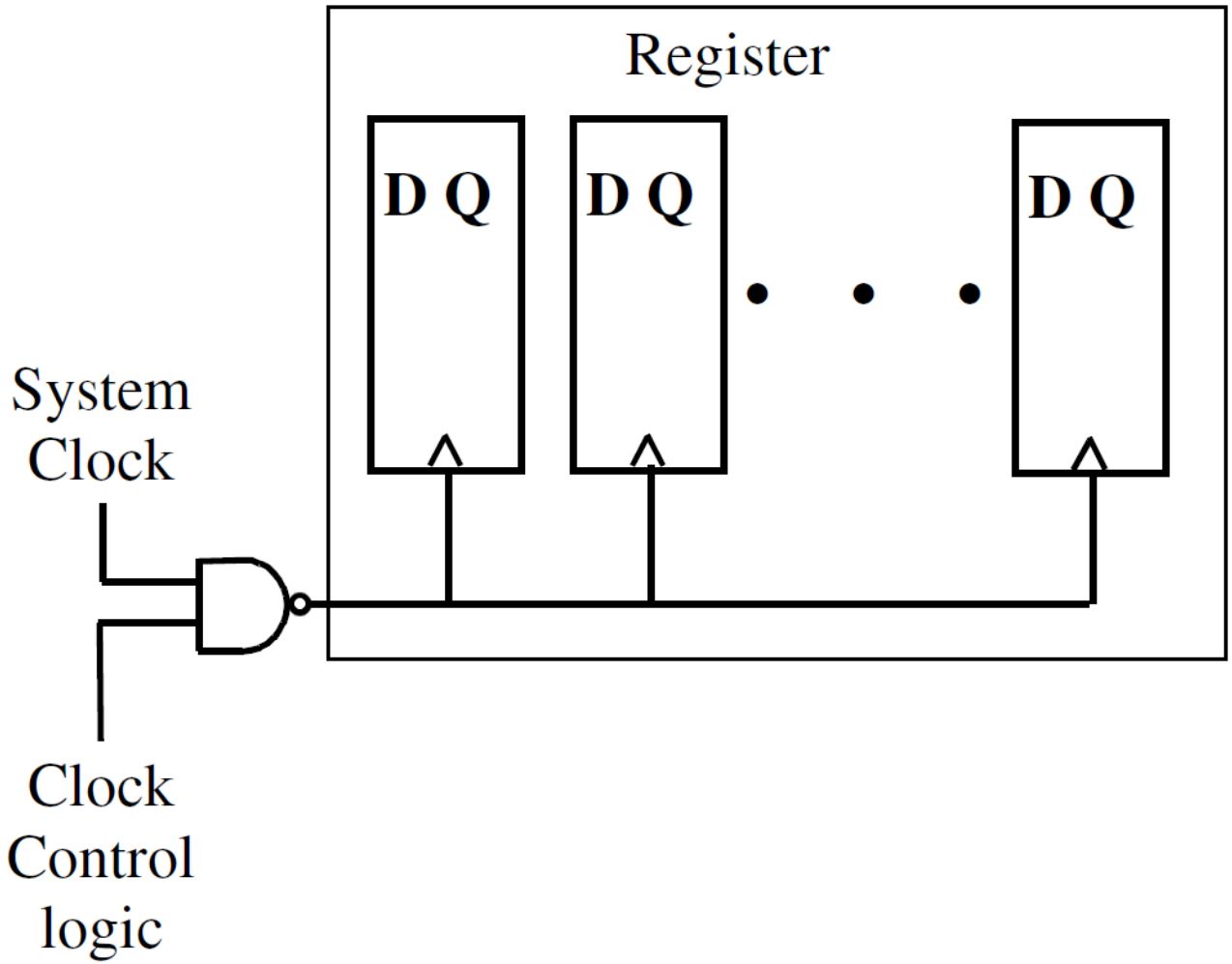




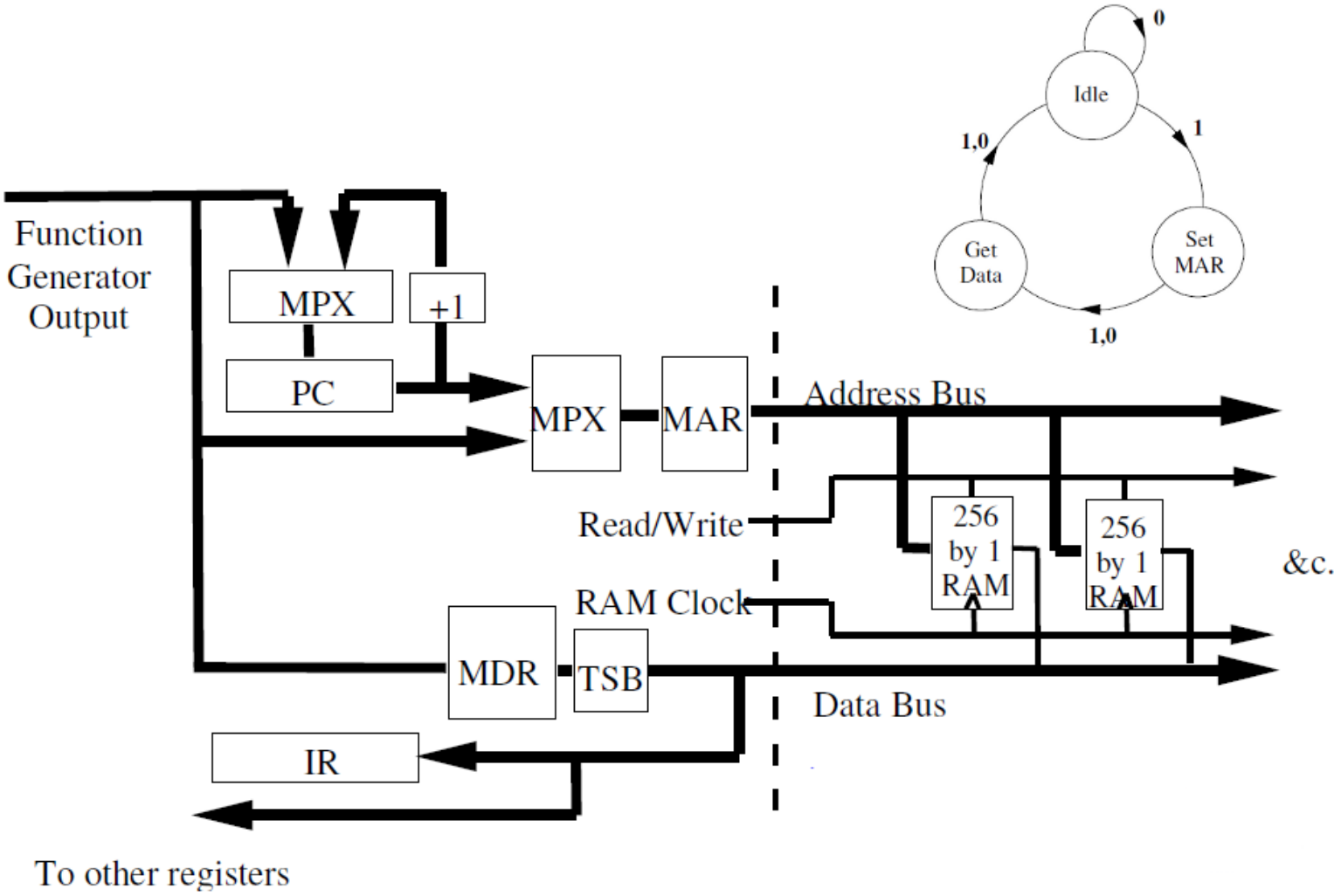
Executing Instructions

1. Set the multiplexers so that the correct registers are connected together
2. Set the arithmetic hardware to compute the correct function
3. Provide a clock falling edge to the registers that are to change.

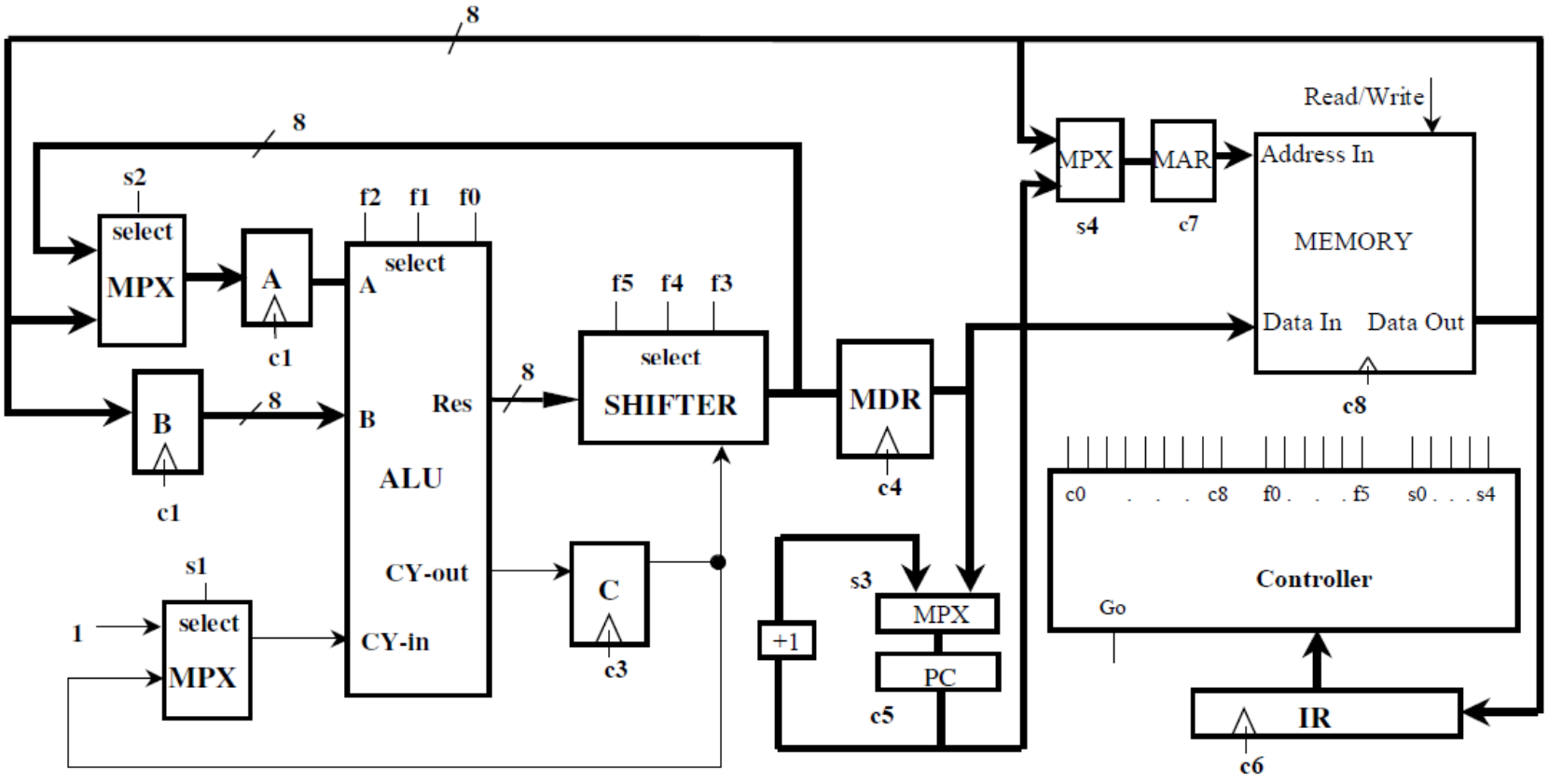
Controlling the individual registers



Fetching from Memory



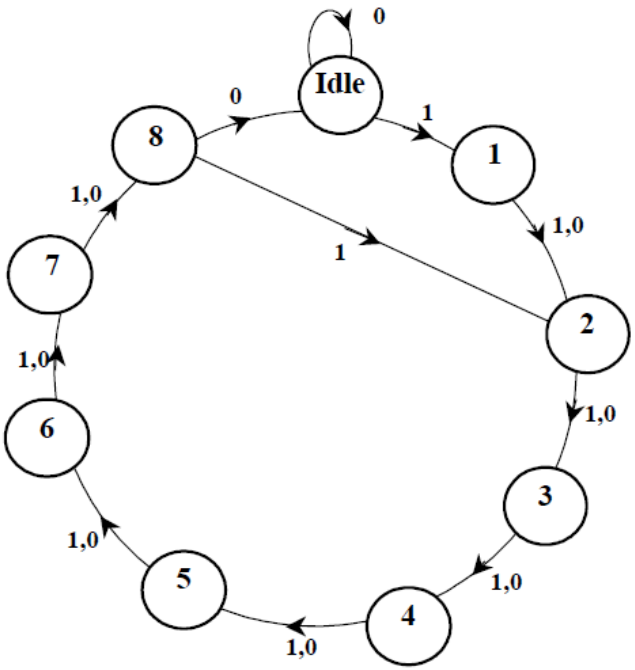
So what about putting the two together?



Changes to the Manual Processor

1. The Result register now becomes the MDR ready to send data to memory
2. MAR can be loaded direct from memory
3. Memory ”data out” separated from ”data in”, and always enabled
4. IR bits now used as input to the controller

A Simple Controller



	Function
0	Wait for the start signal
1	Set address to get the next byte
2	Load the instruction Set address to get the next byte
3	Load data to register A Set address to get the next byte
4	Load data to registers B and C Set address to get the next byte
5	Load the instruction
6	Load data to registers C and MDR Set address to get the next byte
7	Load result address to MAR
8	Store the result Set address to get the next byte

Have we done the job?

Unfortunately not:-

We will never sell a processor that:

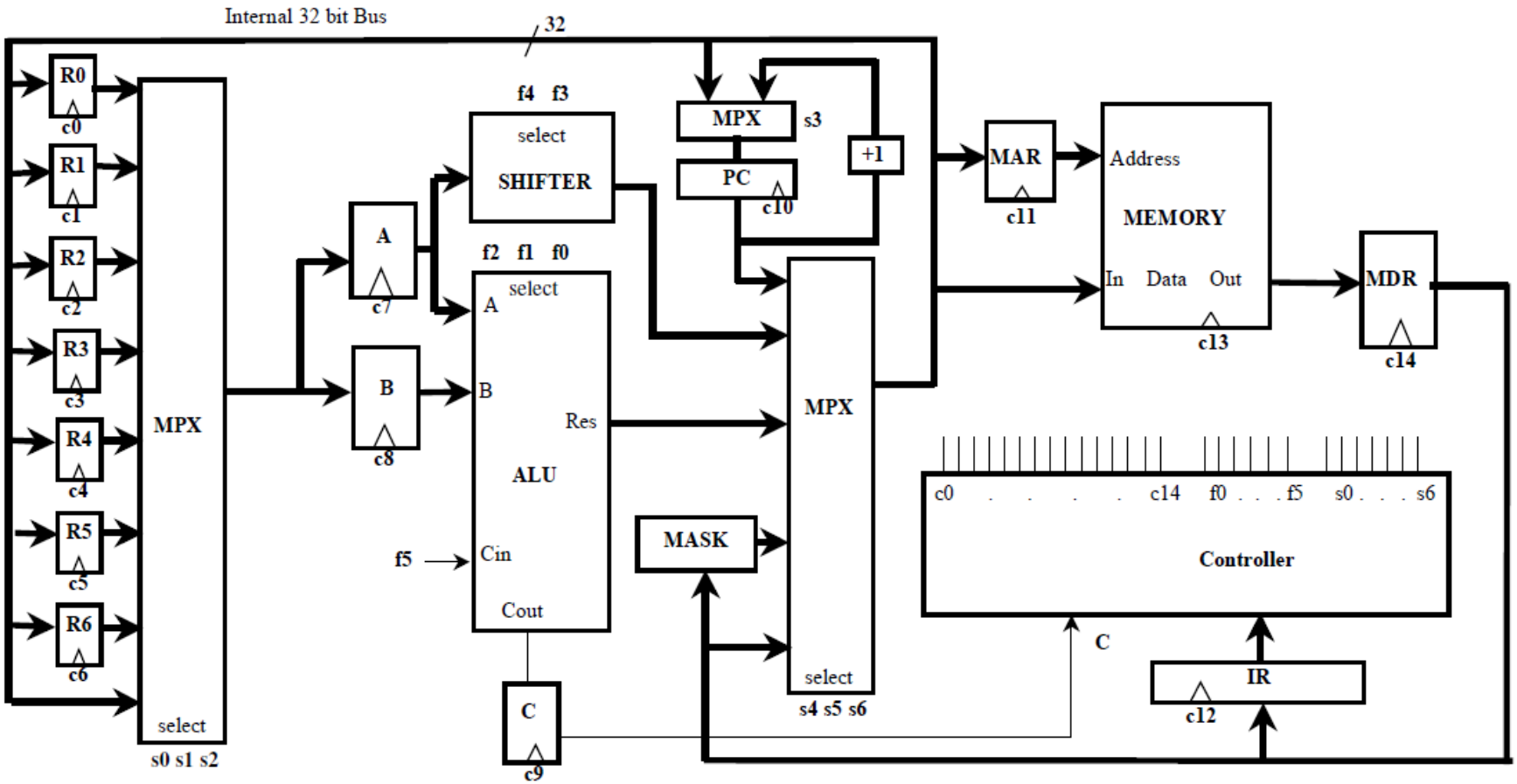
- Continually needs to read and write to memory
- Has a large number of execution cycles per instruction
- Processes only integers up to 255
- Has cumbersome 5 byte instructions

We need to find ways of speeding things up.

How can we speed things up?

1. Change all the buses from 8 bits to 32 bit, and thus do four times as much in each cycle.
2. Provide local registers which can be programmed to store partial results.
3. Reduce the size of the combinatorial logic by putting the ALU and shifter in parallel
4. Design a controller with as small a number of execution cycles as possible.
5. Remove elaborate carry arrangements, by doing our arithmetic on big integers

A Real Processor at Last



Problem

If we wanted to exchange the contents of register R0 and Register R1, what operations must our processor carry out?

Problem

If we wanted to exchange the contents of register R0 and Register R1, what operations must our processor carry out?

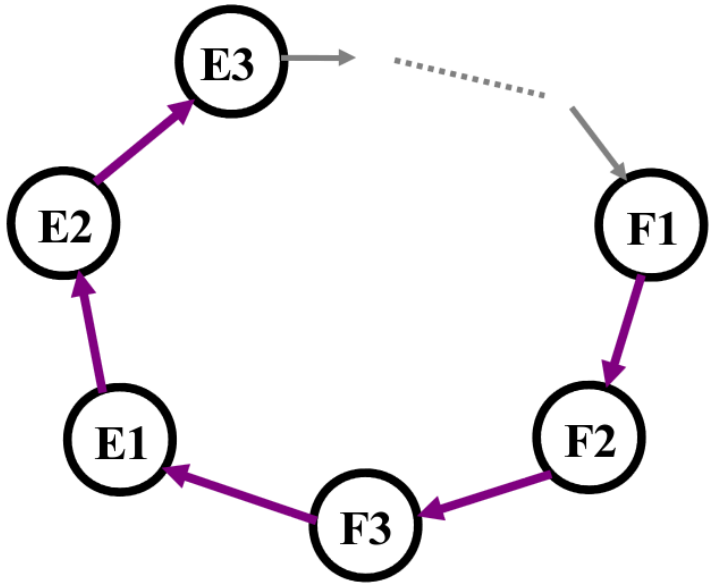
$A \leftarrow R0$	Load A from R0
$R1 \leftarrow A$, $A \leftarrow R1$	Load R1 from A and A from R1
	(at the same time!)
$R0 \leftarrow A$	Load R0 from A

A few observations

- Registers A, B, PC, MAR, MDR, IR and C belong to the hardware designers.
- Registers R0, R1, R2, R3, R4, R5 and R6 can be manipulated by the programmers.
- Most arithmetic operations will no longer require a carry in, so we can set the ALU Cin directly from the controller if required (eg for an increment instruction)

The controller specification

The controller is going to be a sequential circuit looking something like this:



During the F states instructions are fetched from memory

During the E states instructions are executed

The Controller Specification

- The fetch cycle will get one 32 bit instruction from the memory.
- The execute cycles will make the processor carry out that instruction.
- So, to formalise our specification we need to define what the instructions will do.

The Program Instructions

Assembler instructions are decided between the software specialists and the hardware designers.

From the software point of view:

Instructions should be few but very powerful

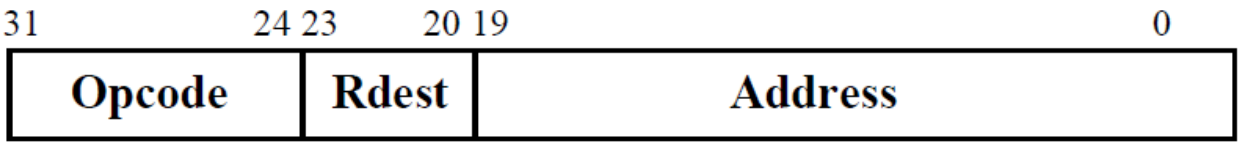
From the hardware point of view:

Instructions should be simple and take only a few clock pulses to execute

A compromise must be found

Instruction Format

- We need to be very precise about our instruction format so that we can easily interpret it in hardware.
- We will assume that there will be 255 or less instructions, and the top eight bits will define the instruction.
- Most instructions will act on a register, so we will let the next four bits define the destination register.

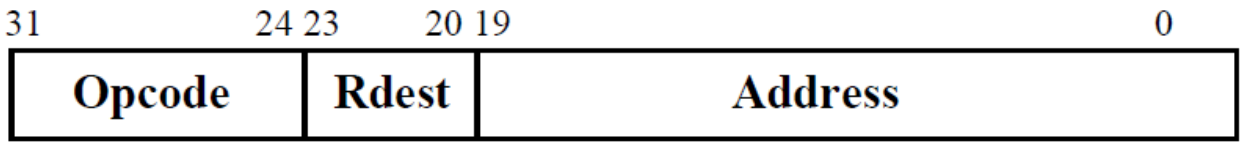


Memory Reference Instructions

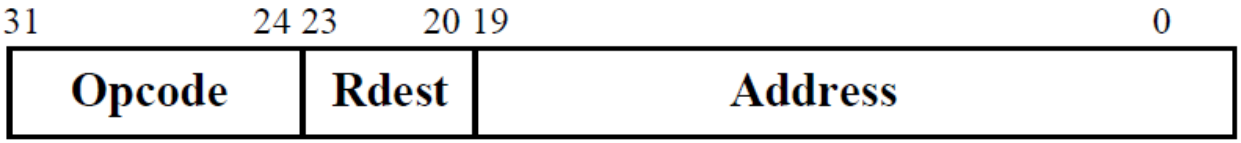
There are four basic instructions that reference the memory directly:

- LOAD Reg, Address
- STORE Reg, Address
- JUMP Address
- CALL Reg, Address

These all have the same instruction format:

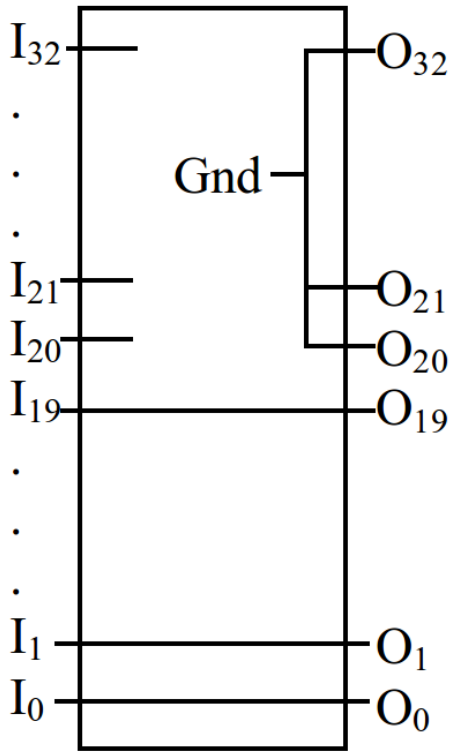


The Function of the Mask is Revealed



The memory reference instructions contain a memory address in the bottom 20 bits.

The mask converts this unsigned 20 bit number to a 32 bit number by masking the top bits.

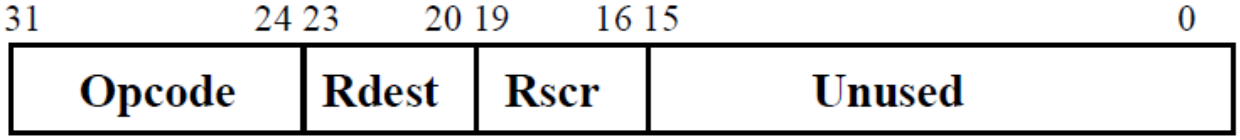


Indirect Memory Reference Instructions

20 bit addresses are a little small - they can only address 4M, so we introduce indirect addressing where a 32-bit address is stored in a register.

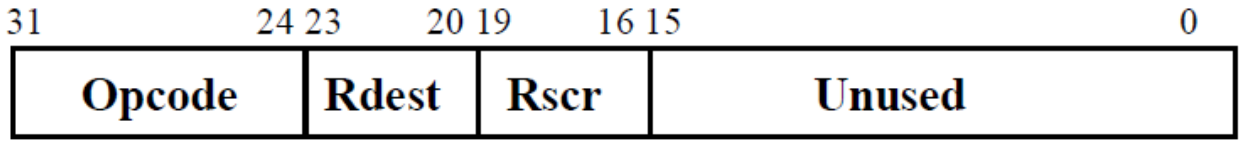
Indirect instructions are slower, but make things easier for the software department.

- LOADINDIRECT Reg1, Reg2
- STOREINDIRECT Reg1, Reg2
- JUMPINDIRECT Reg
- CALLINDIRECT Reg1, Reg2



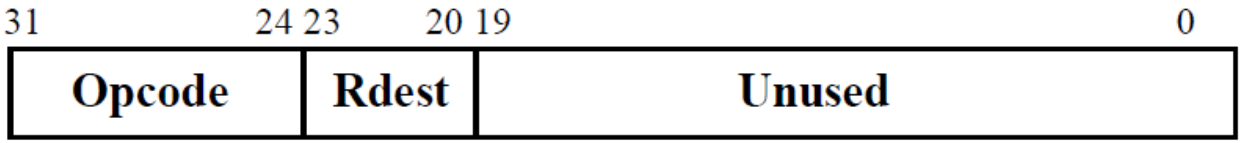
Two register internal instructions

- MOVE Rdest, Rscr
- ADD Rdest, Rscr
- SUBTRACT Rdest, Rscr
- AND Rdest, Rscr
- OR Rdest, Rscr
- XOR Rdest, Rscr
- COMPARE Rdest, Rscr
  (Subtract but without storing the result)



One Register Instructions

- CLEAR Rdest
- INCREMENT (INC Rdest)
- DECREMENT (DEC Rdest)
- COMPLEMENT (COMP Rdest)
- ARITHMETIC SHIFT LEFT (ASL Rdest)
- ARITHMETIC SHIFT RIGHT (ASR Rdest)
- ROTATE RIGHT (ROR Rdest)
- RETURN



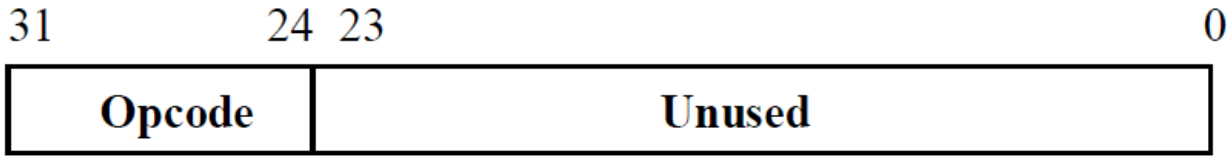
No Register Instructions

Following an arithmetic or compare operation:

- SKIPPOSITIVE
- SKIPNEGATIVE

Unconditional:

- SKIP
- NOP



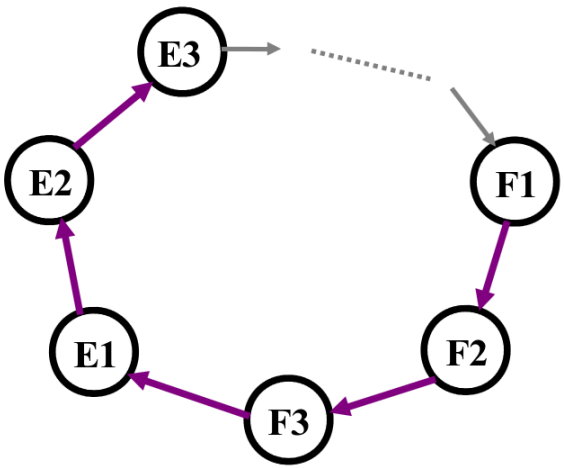
Input and Output

- Input and output will be achieved using the LOAD and STORE instructions.
- Many peripherals, disks, serial ports, etc can be read or written to as if they were memory.
- They can also be connected directly with the memory so that input and output can be done while the processor is otherwise engaged.

Register transfers

- Each of the instructions that we have specified requires a series of register transfers to carry them out.
- We continue our design, by determining the register transfers that will be needed to carry out each instruction.
- These register transfers will become the formal specification of the processor controller.

Register Transfers and processor operation



At each step of the execution cycle a number of register transfers take place.

In the fetch steps the register transfers are always the same.

In the execute steps the register transfers depend on the instruction being executed.

Fetch Register Transfers

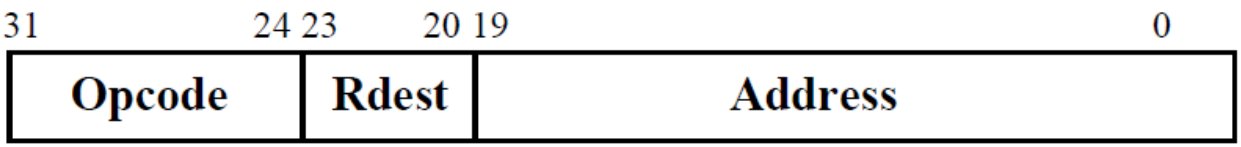
In this version of the processor the MDR is the only register connected to the memory data out bus. Three states are required to fetch an instruction:

- F1: MAR←PC; PC←PC+1
- F2: MDR←Memory
- F3: IR←MDR

After F3 has finished the instruction is in both IR and MDR

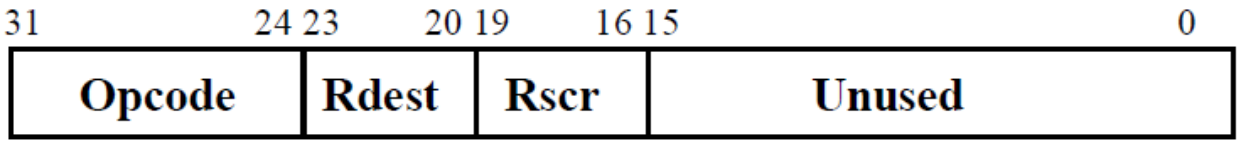
Memory Reference Instructions

Instruction	Cycle	Transfers	Path
LOAD Rdest, Address	E1	MAR←MDR	Use the mask
	E2	MDR←Memory	
	E3	Rdest←MDR	No Mask
STORE Rdest, Address	E1	MAR←MDR; A←Rdest	Via the mask
	E2	Memory←A	Shifter (unchanged)
JUMP Address	E1	PC←MDR	Via the mask
CALL Rdest, Address	E1	PC←PC+1	
	E2	Rdest←PC	
	E3	PC←MDR	Via the mask



Indirect Memory Reference Instructions

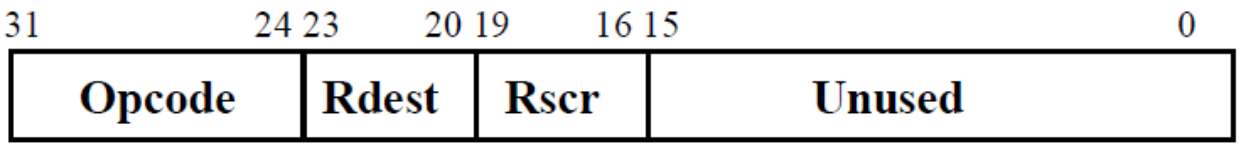
Instruction	Cycle	Transfers	Path
LOADINDIRCET Rdest, Rscr	E1	A←Rsrc	
	E2	MAR←A	
	E3	MDR←Memory	
	E4	Rdest←MDR	No Mask
STOREINDIRECT Rdest, Rscr	E1	A←Rscr	
	E2	MAR←A;A←Rdest	via the shifter
	E3	Memory←A	via the shifter
JUMPINDIRECT Rscr	E1	A←Rscr	
	E2	PC←A	via the shifter
CALLINDIRECT Rdest, Rscr	E1	PC←PC+1;A←Rscr	
	E2	Rdest←PC;	
	E3	PC←A	via the shifter



Two Register Instructions

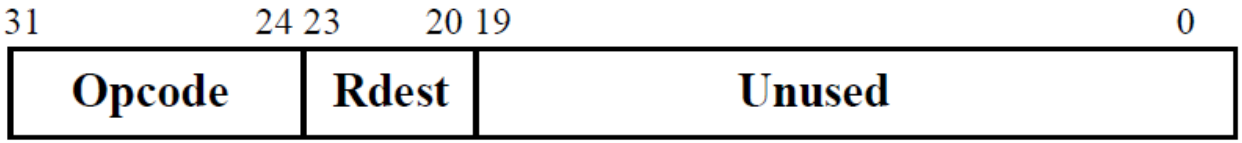
Instruction	Cycle	Transfers	Path
MOVE Rdest, Rsrc	E1	A←Rscr	
	E2	Rdest←Shifter	Shifter (unchanged)
ADD Rdest, Rsrc	E1	A←Rscr	
	E2	B←Rdest	
	E3	Rdest←ALUres; C←ALUcout	ALU=A+B, Cin=0
COMPARE Rdest, Rsrc	E1	A←Rscr	
	E2	B←Rdest	
	E3	C←ALUcout	ALU=A-B, Cin=0

(Plus similar arithmetic and logical operations)



One Register Instructions

Instruction	Cycle	Transfers	Path
CLEAR Rdest	E1	Rdest←ALUres	ALU = zero out
INC Rdest	E1 E2 E3	A←Rdest B←ALUres Rdest←ALUres; C←ALUcout	 ALU = zero out ALU=A+B, Cin=1
DEC Rdest	E1 E2 E3	A←Rdest B←ALUres Rdest←ALUres; C←ALUcout	 ALU = -1 out ALU=A+B, Cin=0
COMP Rdest	E1 E2 E3	A←Rdest B←ALUres Rdest←ALUres	 ALU = -1 out ALU=A eor B
ASL Rdest	E1 E2	A←Rdest Rdest←Shifter	 Shifter (Arithmetic left)
RETURN Rdest	E1 E2	A←Rdest PC←Shifter	 Shifter (Unchanged)

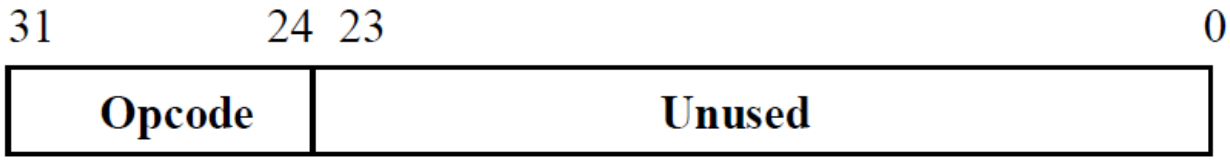


No Register Instructions

Instruction	Cycle	Transfers	Path
SKIP	E1	PC←PC+1	

The conditional skip instructions (SKIPPOSITIVE, SKIPNEGATINE) also take just one execution cycle and may increment the program counter depending on the arithmetic carry.

The NOP instruction doesn't need even one execute cycle, and has no associated register transfers.



To Be Continued

Don't miss the final thrilling episode!