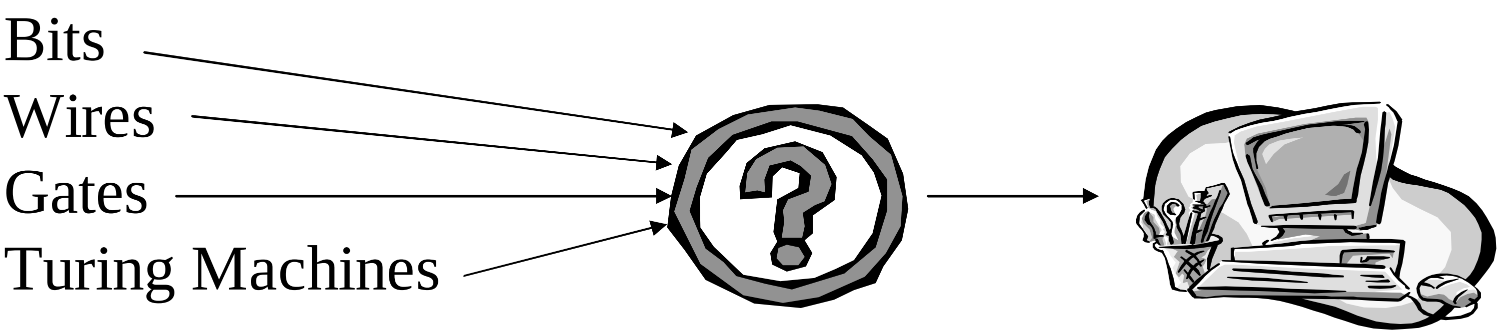
Practical Universal Computers



Lecture 4

Prof. Bienvenido Velez

Outline

- The von Neumann Architecture
  - Big Ideas:
    - Interpretation
    - Stored program concept
- Designing a simple processor
  - Instruction Set Architecture
  - Data Paths
  - Control Unit

The (John) Von Neumann Architecture
(late 40’s)


von Neumann, 1950's

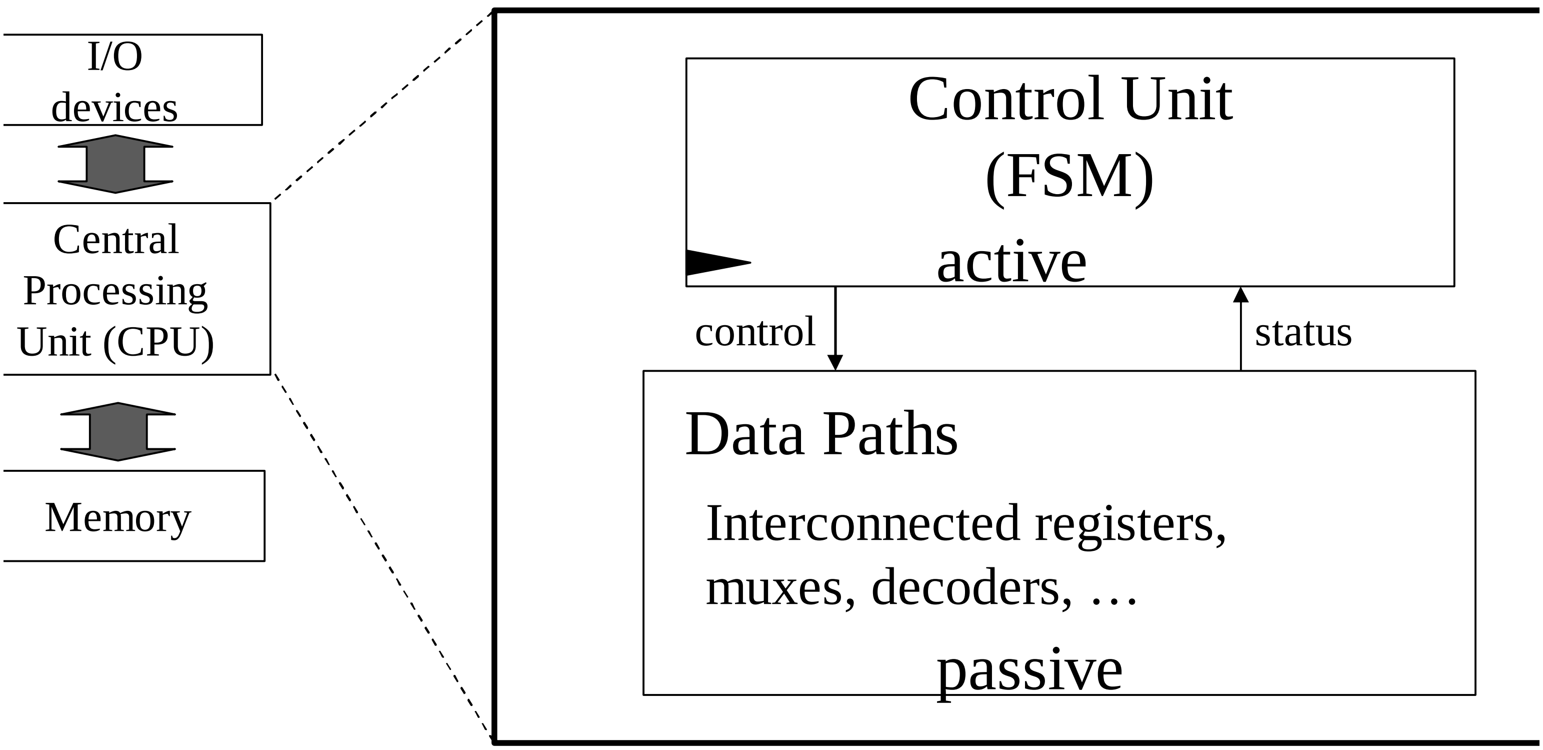
I/O devices — Allow communication with outside world

Central Processing Unit (CPU) — Interprets instructions

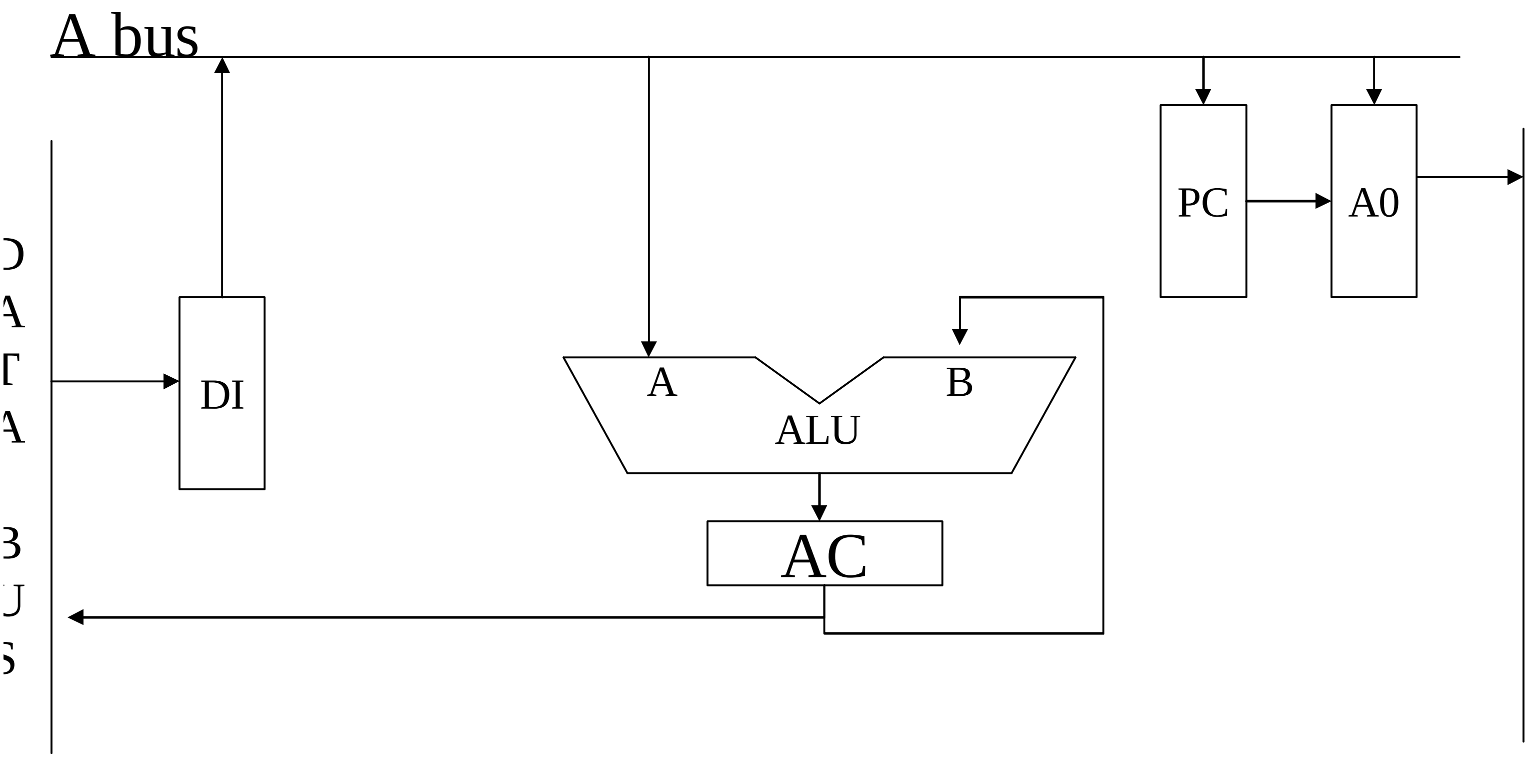
Memory — Stores both programs and

After 60 years … most processors still look like this!

The von Neumann Architecture
Central Processing (CPU)

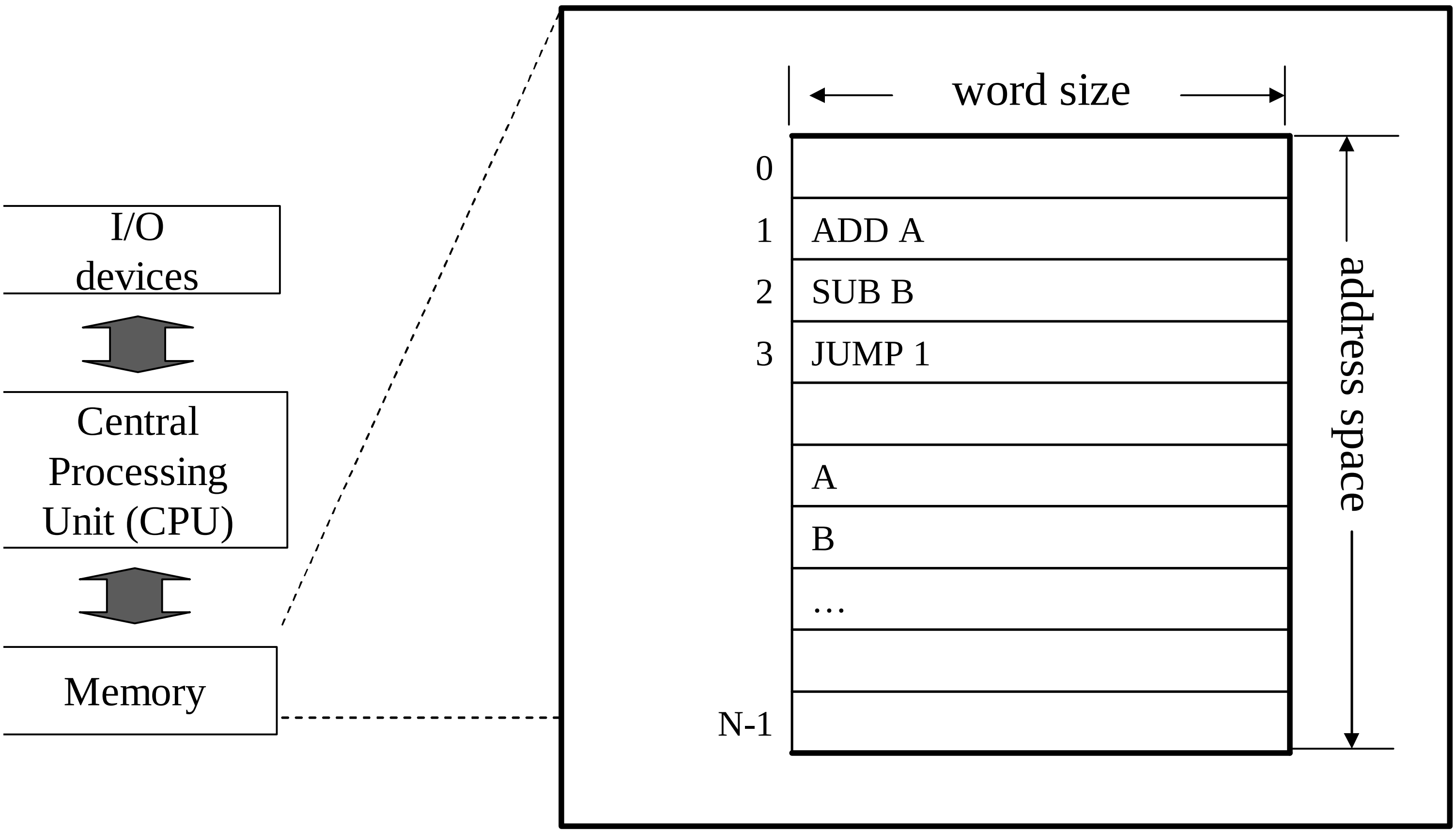


Easy I
Data Paths

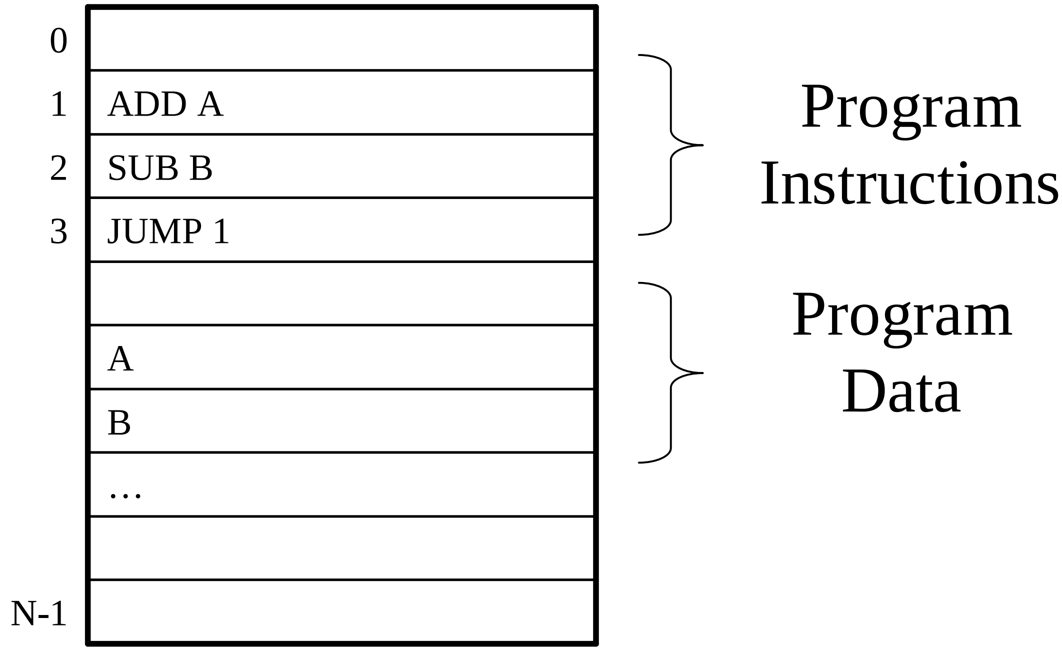


Typically, designing a processor is an iterative (aka trial end error) process

The (John) Von Neumann Architecture
The Memory Unit



The (John) Von Neuman Architecture Stored Program Concept



- Programs and their data coexist in memory
- Processor, under program control, keeps track of what needs to be interpreted as instructions and what as data.

Definition
Instruction Set Architecture

What it is:

– The programmers view of the processor

– Visible registers, instruction set, execution model, memory model, I/O model

What it is not:

– How the processors if build

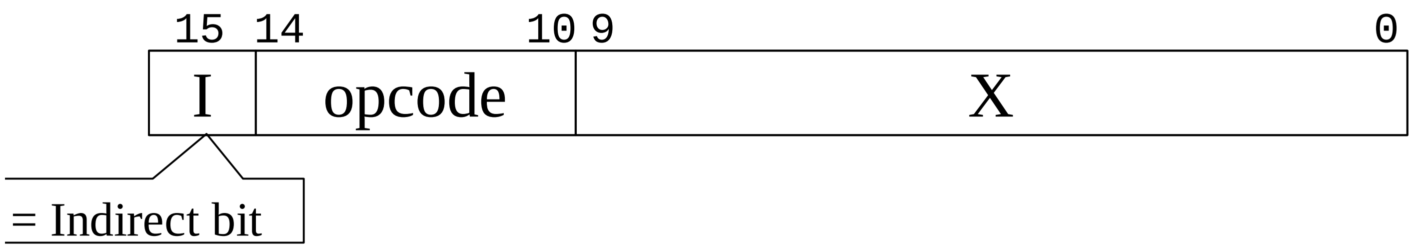
– The processors internal structure

Easy I

A Simple Accumulator Processor

Instruction Set Architecture (ISA)

Instruction Format (16 bits)



Easy I

A Simple Accumulator Processor

Instruction Set Architecture (ISA)

Instruction Set

Name	Opcode	Action I=0	Action I=1
Comp	00 000	AC ? not AC	AC <- not AC
ShR	00 001	AC ? AC / 2	AC ? AC / 2
BrN(i)	00 010	AC < 0 ? PC ? X	AC < 0 ? PC ? MEM[X]
Jump(i)	00 011	PC ? X	PC ? MEM[X]
Store(i)	00 100	MEM[X] ? AC	MEM[MEM[X]] ? AC
Load(i)	00 101	AC ? MEM[X]	AC ? MEM[MEM[X]]
And(i)	00 110	AC ? AC and X	AC ? AC and MEM[X]
Add(i)	00 111	AC ? AC + X	AC ? AC + MEM[X]

Easy all right … but universal it is!

Easy I
Memory Model

	← 8 bits →	← 8 bits →
0		
2	ADD A	
4	SUB B	
6	JUMP 1	
	A	
	B	
	…	
512		

Easy I

A Simple Accumulator Processor

Instruction Set Architecture (ISA)

Some Immediate Observations on the Easy I ISA

Accumulator (AC) is implicit operand to many instructions. No need to use instruction bits to specify one of the operands. More bits left for address and opcodes.

Although simple, Easy I is universal. (given enough memory). Can you see this?

Immediate bit specifies level of indirection for the locatior of the operand. I = 0: operand in X field (immediate). I=1 operand in memory location X (indirect).

Programming the Easy I

Compute the sum of the even numbers between 1 and N

[Blackboard]

Programming the Easy I

Compute the sum of the even numbers between 1 and N

High Level Language Version

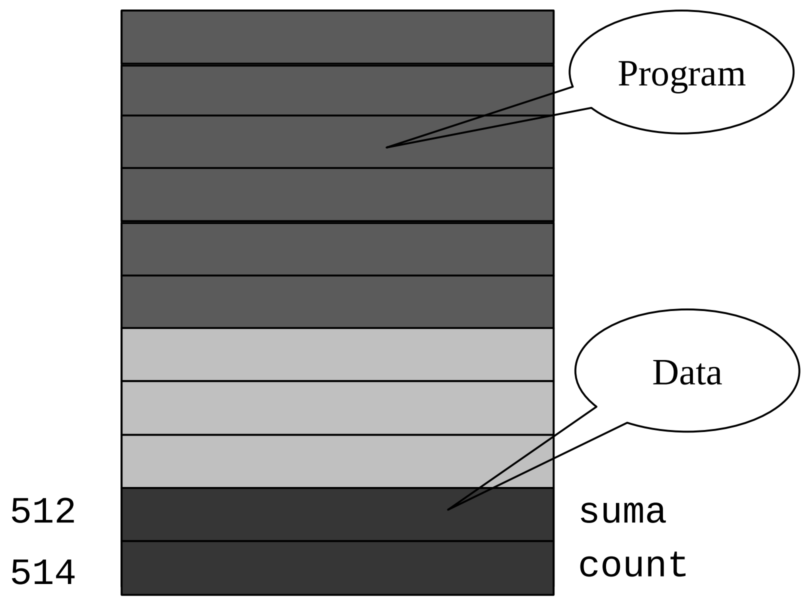
```
int suma = 0;
Int count = 0;
For (count=2; count <= N; count += 2) {
    suma += count;
}
```

Programming the Easy I

- Compute the sum of the even numbers between 1 and N

Easy I Assembly Language Version

Easy I Memory



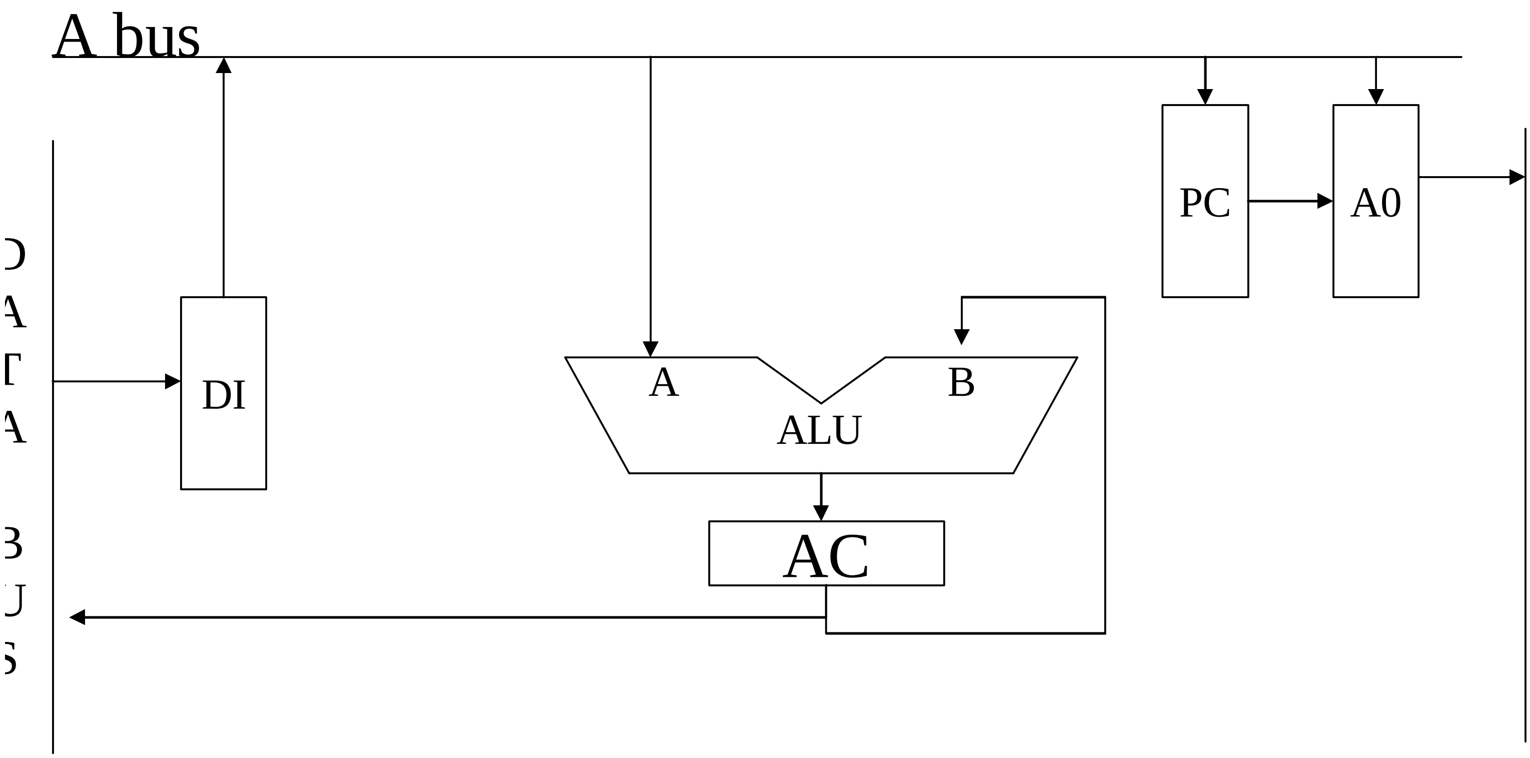
ompute the sum of even numbers from 2 to

Easy I Assembly Language & Machine Code Versions

uction ress	Assembly Code			Comment	Machine Coc (binary)
00		ANDi	0		0 00110 000000
02		STOREi	512	suma = 0	0 00100 100000
04		ADDi	2	count = 2	0 00111 000000
06		STOREi	514		0 00100 100000
08	LOOP:	LOADi	514	for(count=2;count<=N;count+=2){	0 00101 100000
10		COMP			0 00000 dddddd
12		ADDi	1	// add 2's comp of count + N	0 00111 000000
14		ADD	N	// N assumed in MEM[516]	1 00111 100000
16		BrNi	END		0 00010 001000
18		LOADi	512		0 00101 100000
20		ADD	514	suma += count;	1 00111 100000
22		STOREi	514		0 00100 100000
24		LOADi	514		0 00101 100000
26		ADDi	2		0 00111 000000
28		STOREi	514		0 00100 100000
30		JUMPi	LOOP	}	0 00011 000110
32	END:				

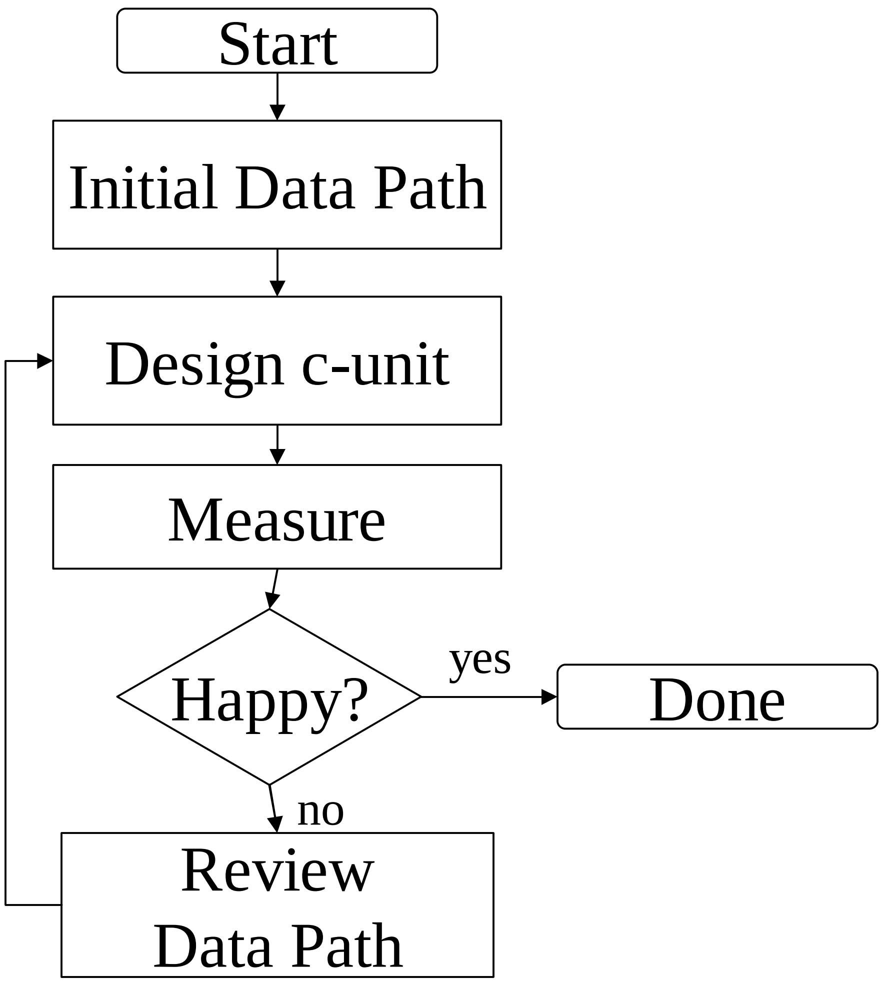
dddddddddd = Don't Care

Easy I
Data Paths

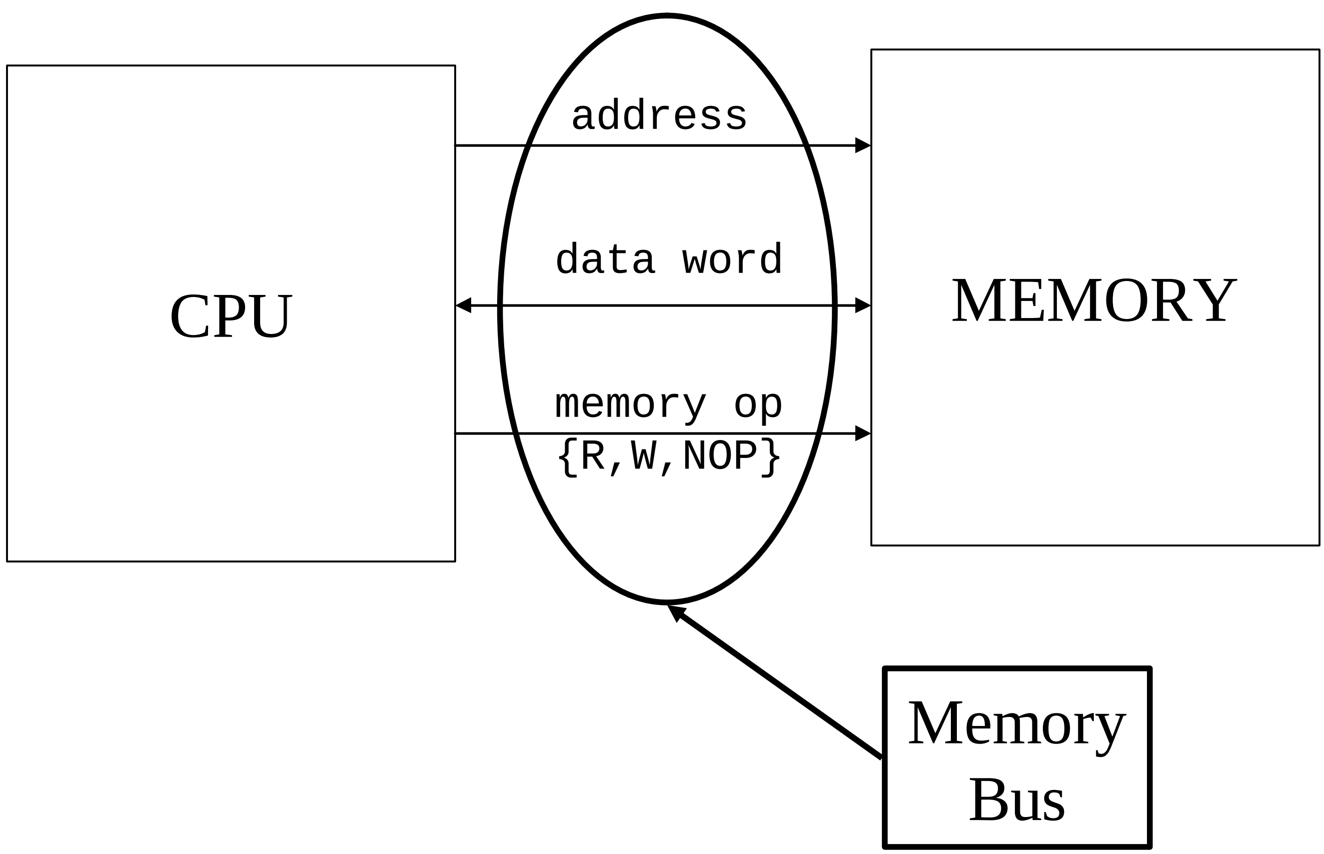


Typically, designing a processor is an iterative (aka trial end error) process

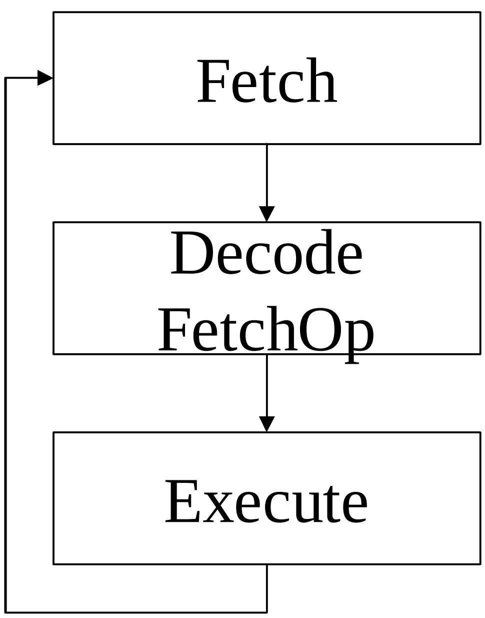
Processor Design Process



Easy I
Memory Interface



Easy I
Control Unit
(Level 0 Flowcharts)



Read next instruction

Determine what it does and prepare to do it. Fetch operands.

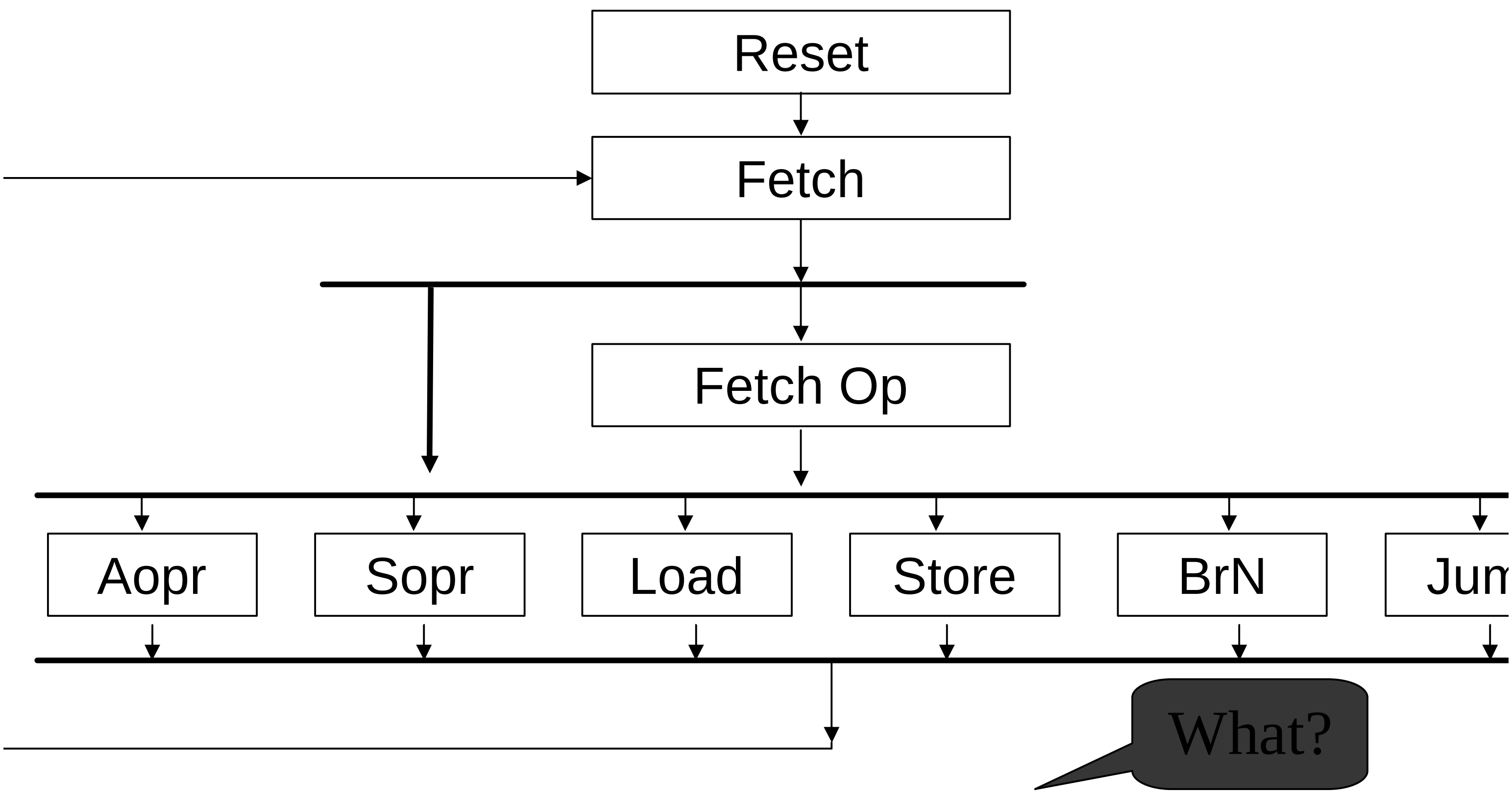
Do it!

We will ignore indirect bit (assuming I = 0) for now

Easy I

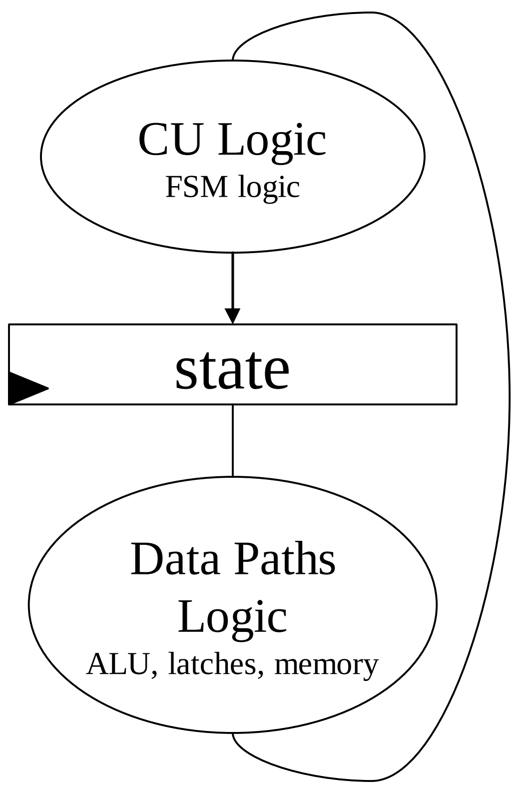
Control Unit

(Level 1 Flowcharts)



Level 1: Each box may take several <u>CPU cycles</u> to execute

What makes a CPU cycle?



Cycle time must accommodate signal propagation

Performance Assessment

IE ~ Instructions executed

CPI ~ Clock cycles per instruction

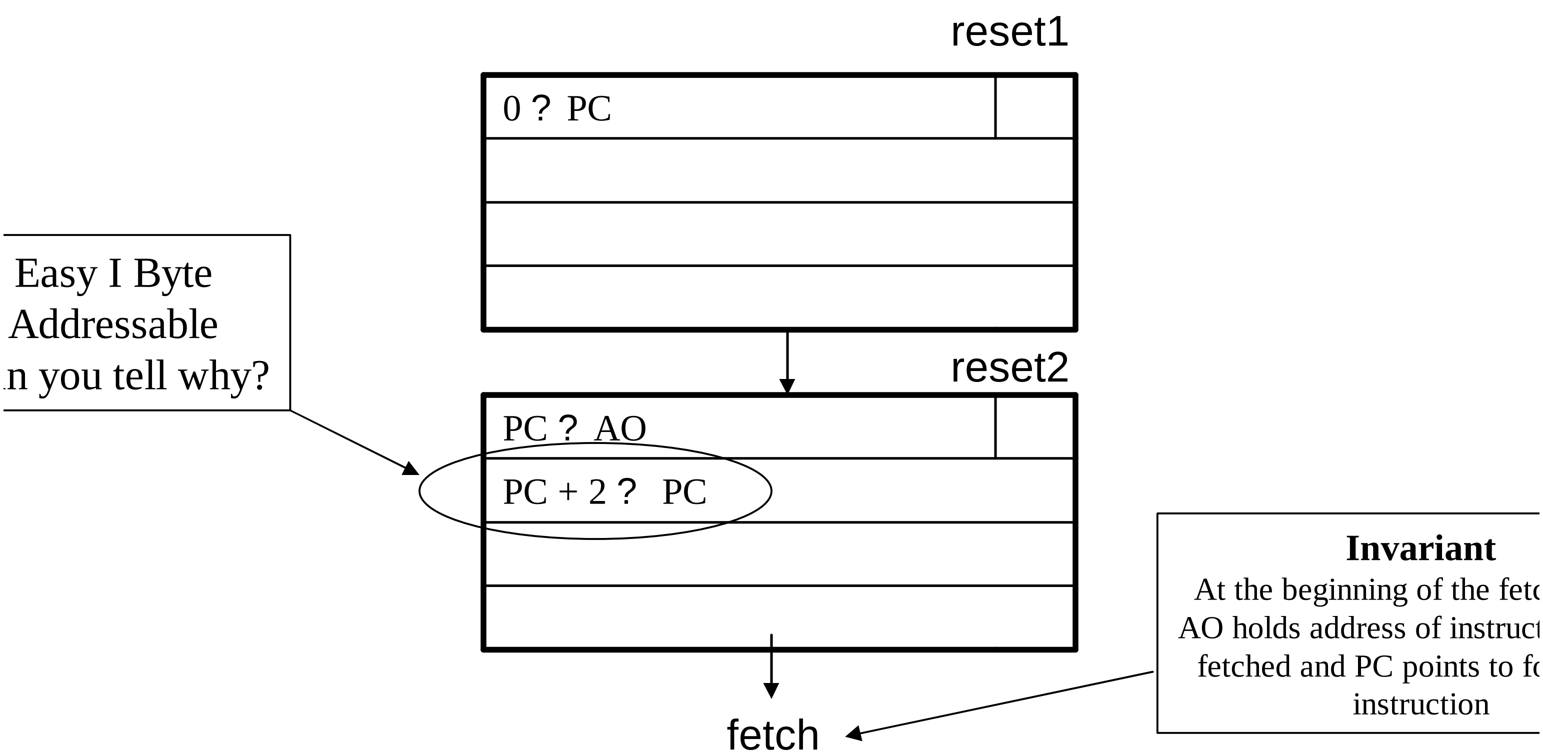
CT ~ Cycle time

Execution time = IE ? CPI ? CT

Key
Performance
Metric

RES

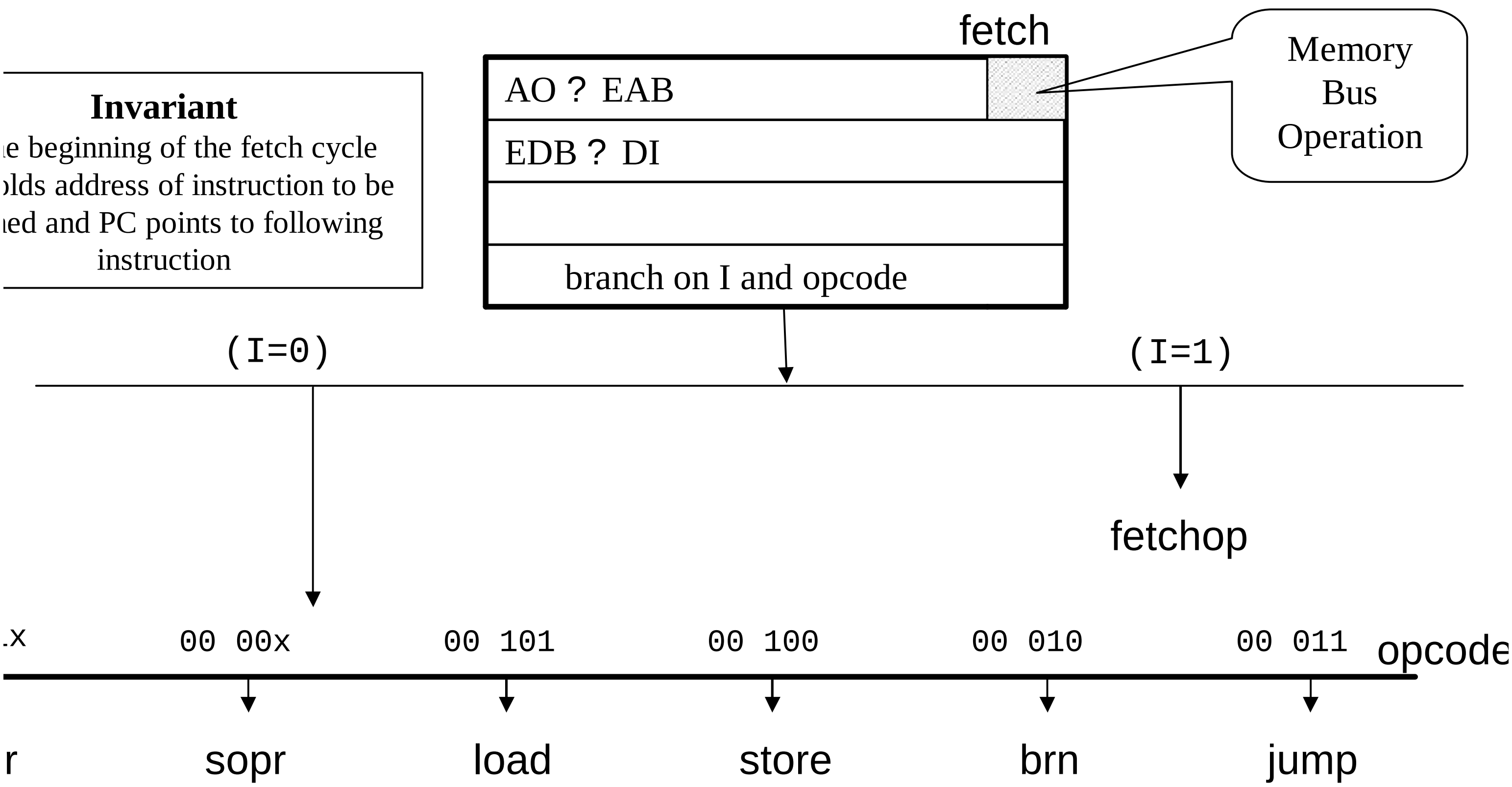
Easy I
Control Unit
(Level 2 Flowcharts)



Each box may take only one CPU cycle to execute

FE

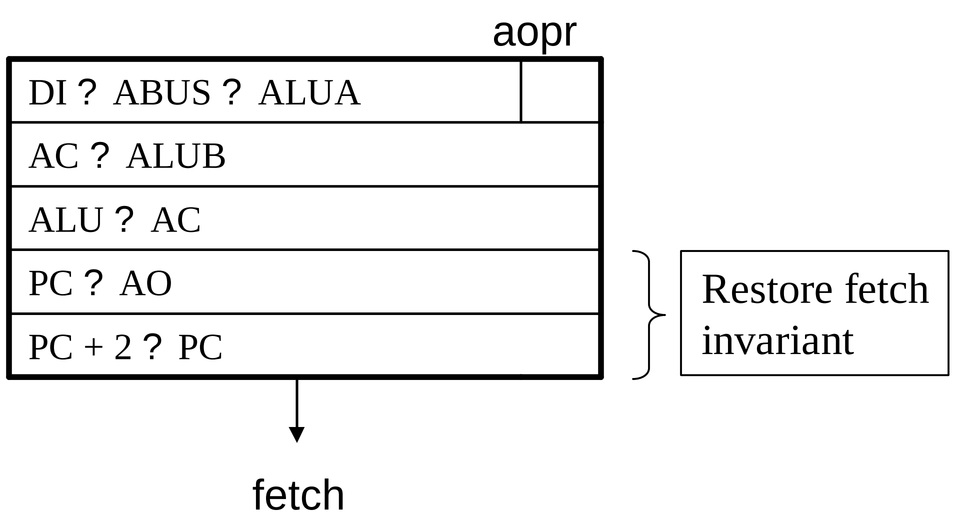
Easy I
Control Unit
(Level 3 Flowcharts)



Opcode must be an input to CU's sequential circuit

AOp

Easy I
Control Unit
(Level 2 Flowcharts)



SOp

Easy I
Control Unit
(Level 2 Flowcharts)

sopr

AC ? ALUB

ALU ? AC

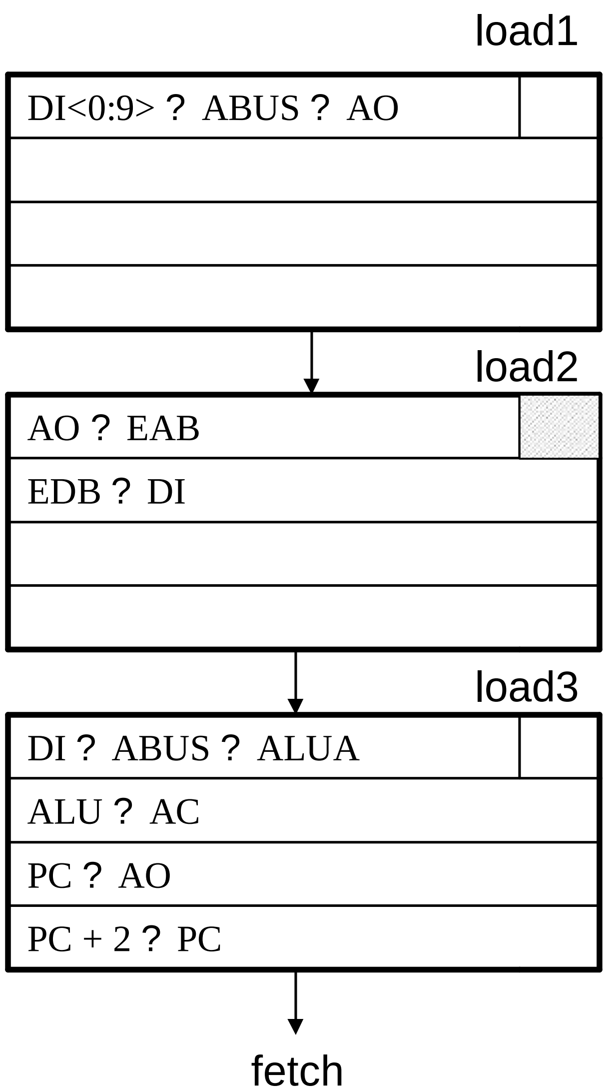
PC ? AO

PC + 2 ? PC

fetch

Loa

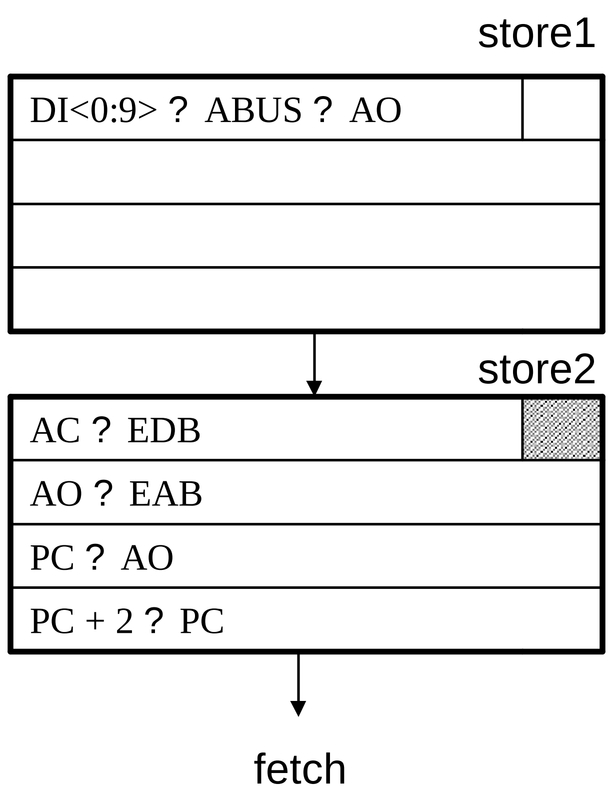
Easy I
ontrol Unit
evel 2 Flowcharts)



Easy I

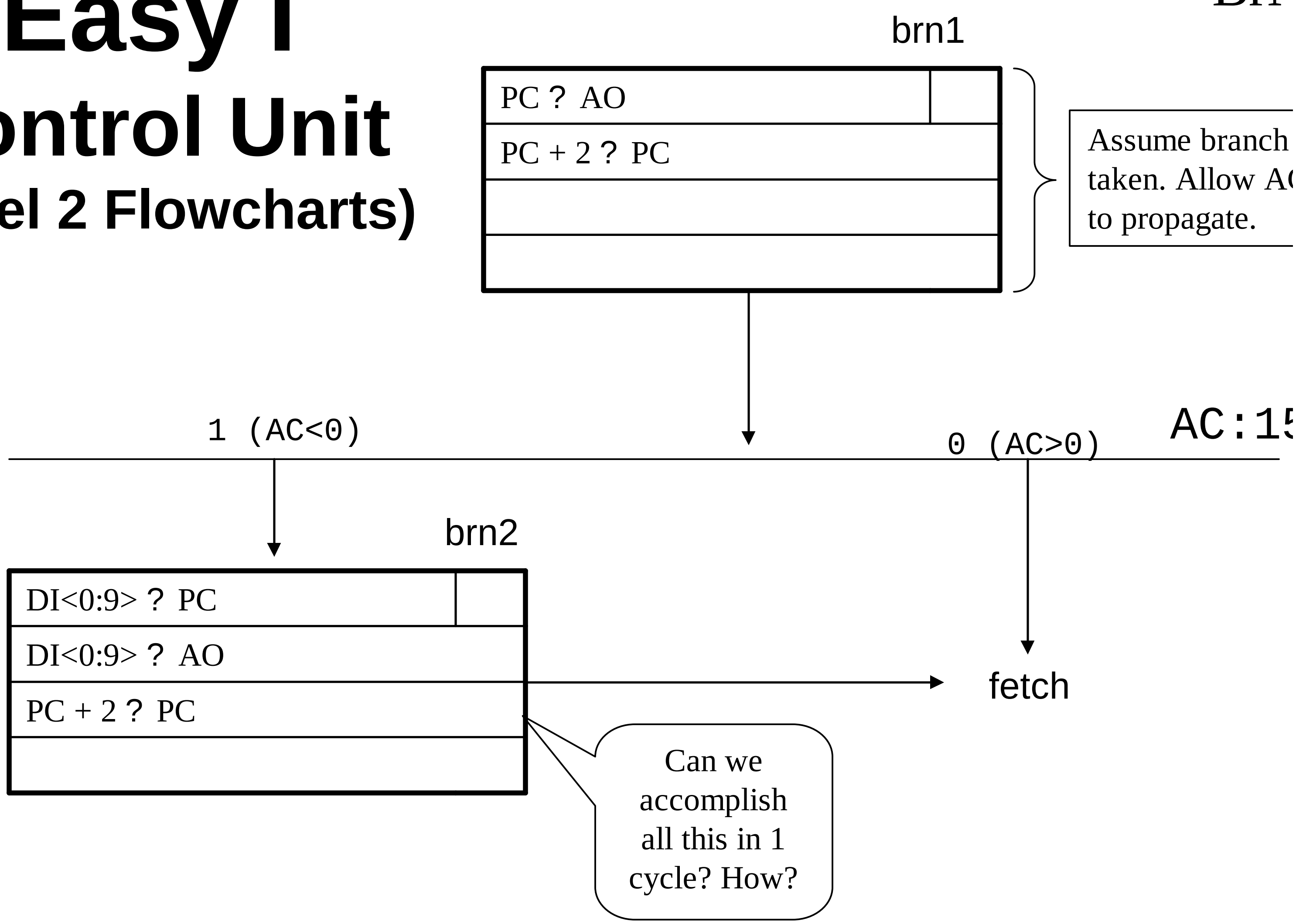
ontrol Unit

evel 2 Flowcharts)



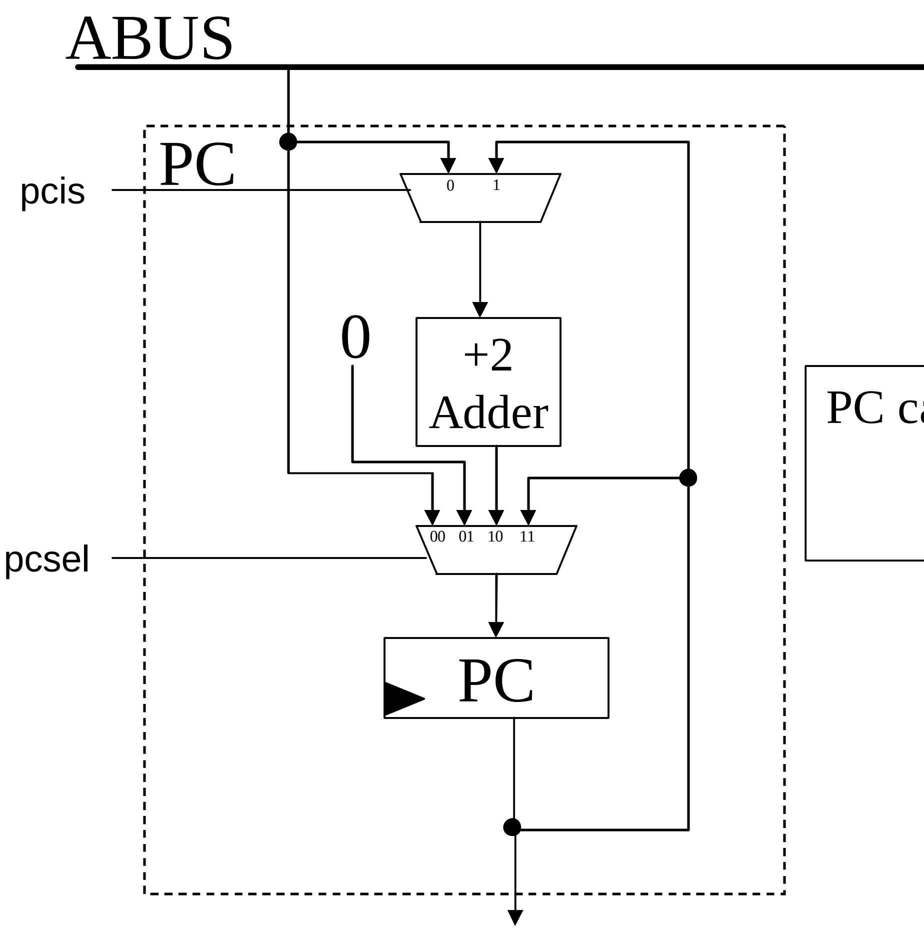
Easy I
ontrol Unit
evel 2 Flowcharts)

BrN



Bit 15 of AC input to the CU's sequential circuit

Inside the Easy-I PC



PC capable of loading ar
incrementing
simultaneously

JUN

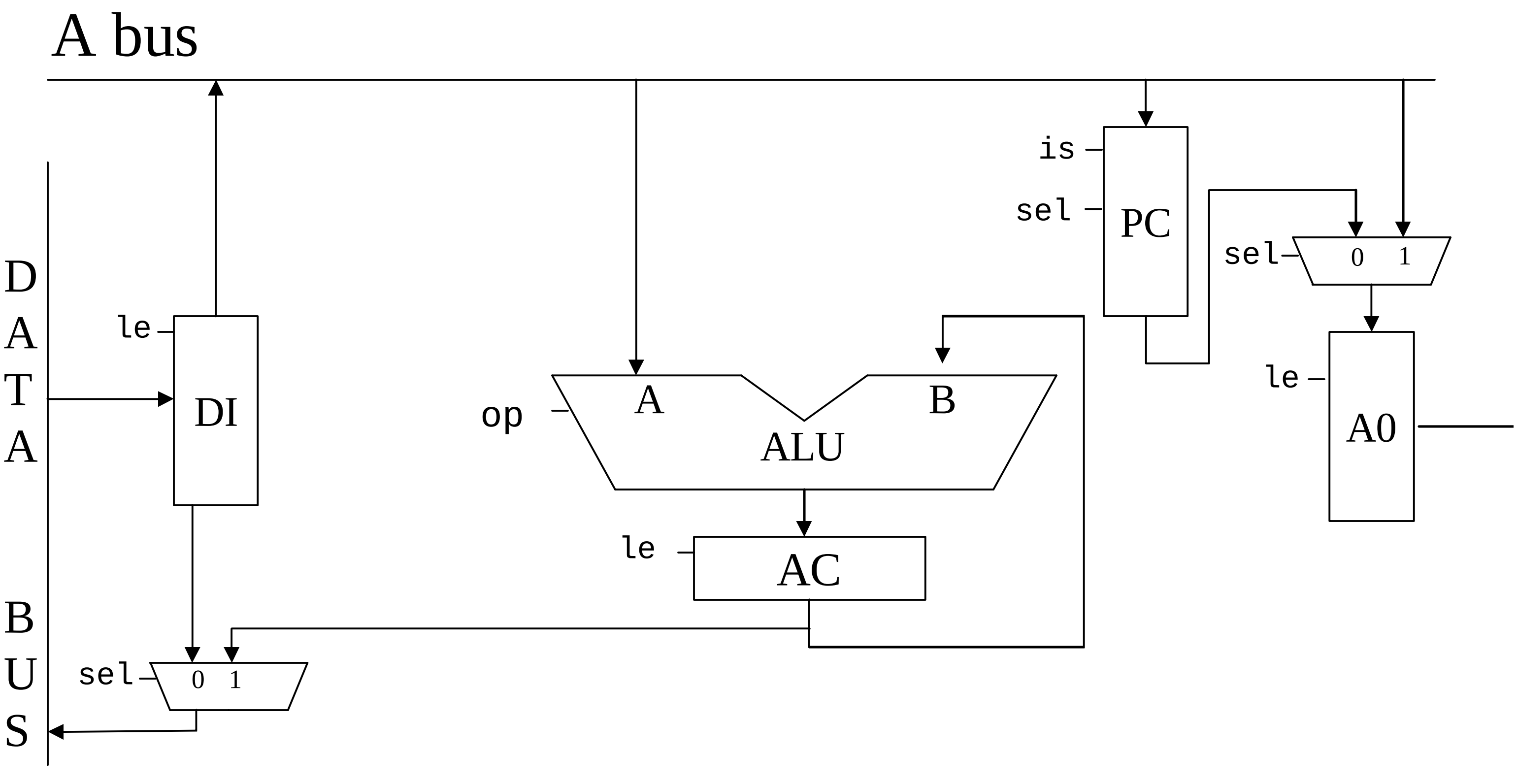
Easy I
Control Unit
(Level 2 Flowcharts)

jump

DI<0:9> ? PC	
DI<0:9> ? AO	
PC + 2 ? PC	

fetch

Easy I
Data Paths (with control points)



Easy I
Control Unit State Transition Table (Part I)

rr te	opcode	I	AC:15		Next State	ALU op	Mem OP	PC sel	PC is	DI le	AC le	AO se l	AO le	
et1	xx xxx	x	x		reset2	XXX	NOP	01	X	0	0	X	0	
et2	xx xxx	x	x		fetch	XXX	NOP	10	1	0	0	0	1	
ch	00 00x	0	x		sopr	XXX	NOP	11	X	1	0	X	0	
ch	00 010	0	x		brn1	XXX	RD	11	X	1	0	X	0	
ch	00 011	0	x		jump	XXX	RD	11	X	1	0	X	0	
ch	00 100	0	x		store1	XXX	RD	11	X	1	0	X	0	
ch	00 101	0	x		load1	XXX	RD	11	X	1	0	X	0	
ch	00 11x	0	x		aopr	XXX	RD	11	X	1	0	X	0	
or	00 110	x	x		fetch	AND	NOP	10	1	0	1	0	1	
or	00 111	x	x		fetch	ADD	NOP	10	1	0	1	0	1	
or	00 000	x	x		fetch	NOTB	NOP	10	1	0	1	0	1	
or	00 001	x	x		fetch	SHRB	NOP	10	1	0	1	0	1	

Easy I
Control Unit State Transition Table (Part II)

opcode	I	AC:15		Next State	ALU op	Mem OP	PC sel	PC is	DI le	AC le	AO sel	AO le
xx xxx	x	x		store2	XXX	NOP	11	X	0	0	1	1
xx xxx	x	x		store3	XXX	WR	10	1	0	0	0	1
xx xxx	x	x		load2	XXX	NOP	11	X	0	0	1	1
xx xxx	x	x		load3	XXX	RD	11	X	1	0	X	0
xx xxx	x	x		fetch	XXX	NOP	10	1	0	1	0	1
xx xxx	x	0		fetch	XXX	NOP	10	1	0	0	0	1
xx xxx	x	1		brn2	XXX	NOP	10	1	0	0	0	1
xx xxx	x	x		fetch	XXX	NOP	10	0	0	0	1	1
xx xxx	x	x		fetch	XXX	NOP	10	0	0	0	1	1

CU with 14 states => 4 bits of state

Easy-I Control Unit – Some missing details

t Encodings for States

State	Encoding
`reset1`	0000
`reset2`	0001
`fetch`	0010
`aopr`	0011
`sopr`	0100
`store1`	0101
`store2`	0110
`store3`	0111
`load1`	1000
`load2`	1001
`load3`	1010
`brn1`	1011
`brn2`	1100
`jump`	1101

ALU Operation Table

Operation	Code	Output
A	000	A
NOTB	001	not B
AND	010	A and B
ADD	011	A + B
SHRB	100	B / 2

We know how to implement this ALU

Control Bus Operation Table

Operation	Code
NOP	00
Rea**D**	01
WRite	10

Easy I
Control Unit State Transition Table (Part I)

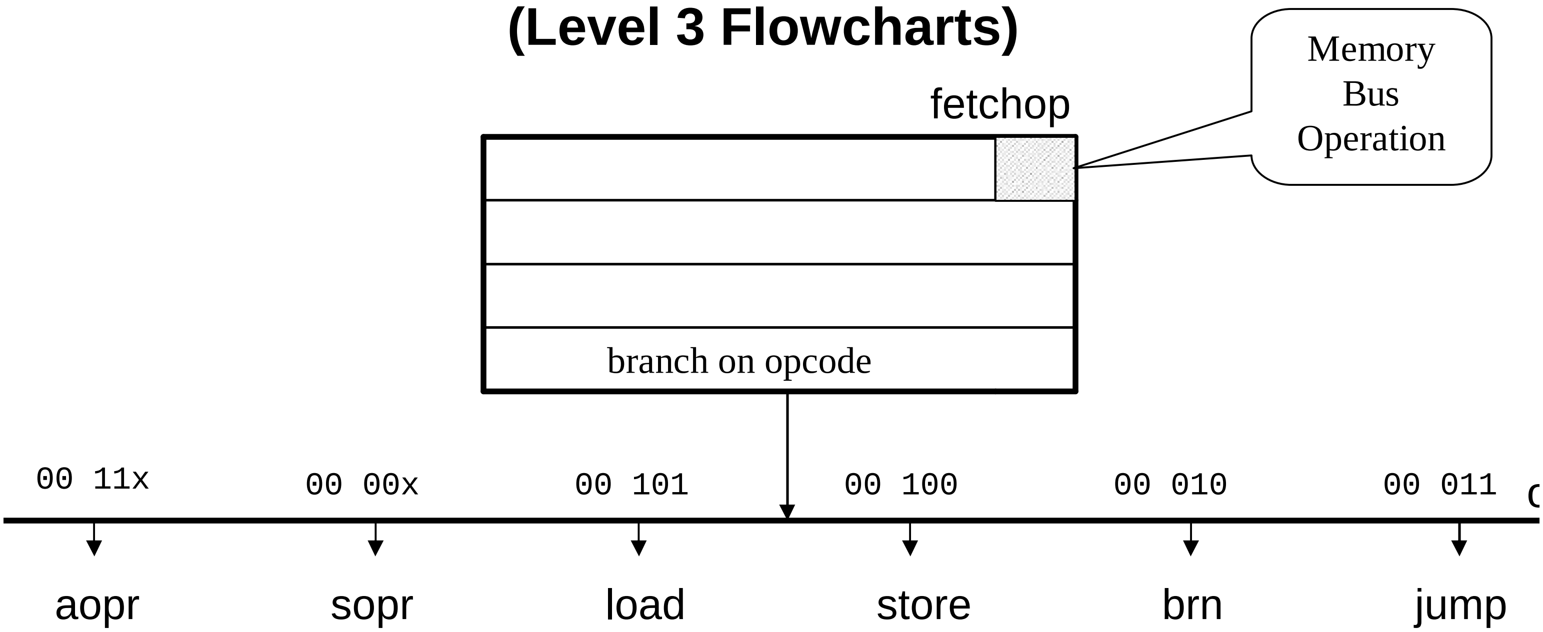
- e	opcode	I	AC: 15		Next State	ALU op	Mem OP	PC sel	PC is	DI le	AC le	AO sel	AO le	
)	xx xxx	x	x		0001	XXX	00	01	X	0	0	X	0	
L	xx xxx	x	x		0010	XXX	00	10	1	0	0	0	1	
)	00 00x	0	x		0100	XXX	00	11	X	1	0	X	0	
)	00 010	0	x		1011	XXX	01	11	X	1	0	X	0	
)	00 011	0	x		1101	XXX	01	11	X	1	0	X	0	
)	00 100	0	x		0101	XXX	01	11	X	1	0	X	0	
)	00 101	0	x		1000	XXX	01	11	X	1	0	X	0	
)	00 11x	0	x		0011	XXX	01	11	X	1	0	X	0	
L	00 110	x	x		0010	010	00	10	1	0	1	0	1	
L	00 111	x	x		0010	011	00	10	1	0	1	0	1	
)	00 000	x	x		0010	001	00	10	1	0	1	0	1	
)	00 001	x	x		0010	100	00	10	1	0	1	0	1	

Easy I

Control Unit State Transition Table (Part II)

ent te	opcode	I	AC:15		Next State	ALU op	Mem OP	PC sel	PC is	DI le	AC le	AO sel	AO le
1	xx xxx	x	x		0110	XXX	00	11	X	0	0	1	1
0	xx xxx	x	x		0111	XXX	10	10	1	0	0	0	1
0	xx xxx	x	x		1001	XXX	00	11	X	0	0	1	1
1	xx xxx	x	x		1010	XXX	01	11	X	1	0	X	0
0	xx xxx	x	x		0010	XXX	00	10	1	0	1	0	1
1	xx xxx	x	0		0010	XXX	00	10	1	0	0	0	1
1	xx xxx	x	1		1100	XXX	00	10	1	0	0	0	1
0	xx xxx	x	x		0010	XXX	00	10	0	0	0	1	1
1	xx xxx	x	x		0010	XXX	00	10	0	0	0	1	1

Easy I
Control Unit
(Level 3 Flowcharts)



Opcode must be an input to CU's sequential circuit

Building the Easy-I C-Unit
2 Approaches

Harwired

– Apply well known sequential circuit techniques

Micro-programmed

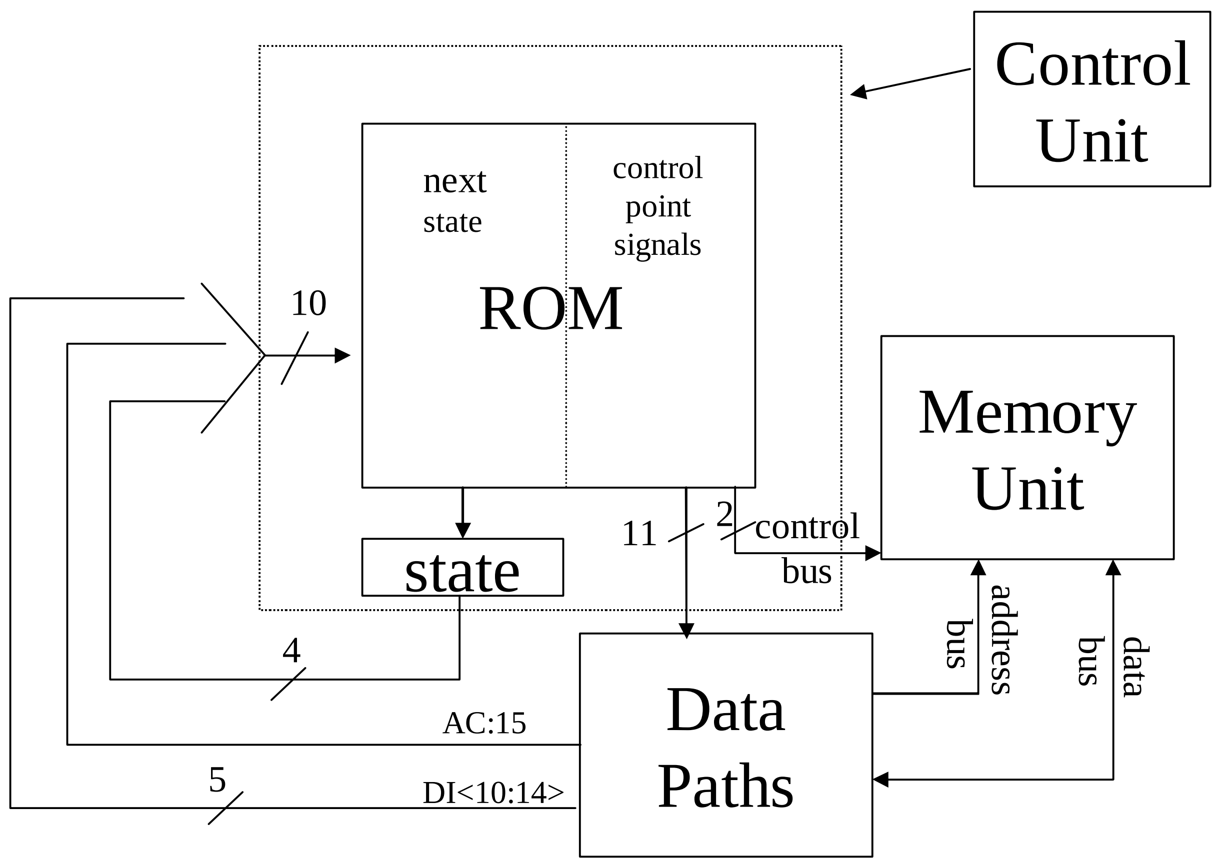
– Treat state transition table as a program

A
? prog

– Build a new abstraction layer

The Microprogramming abstraction level

Building the Easy-I C-Unit
Hardwired Approach



Easy I

Control Unit State Transition Table (Part II)

t	opcode	AC:15		Next State	ALU op	Mem OP	PC sel	PC is	DI le	AC le	AO sel	AO le
	xx xxx	x		0110	XXX	000	11	X	0	0	1	1
	xx xxx	x		0111	XXX	010	10	1	0	0	0	1
	xx xxx	x		1001	XXX	000	11	X	0	0	1	1
	xx xxx	x		1010	XXX	001	11	X	1	0	X	0
	xx xxx	x		0010	XXX	000	10	1	0	1	0	1
	xx xxx	0		0010	XXX	000	10	1	0	0	0	1
	xx xxx	1		1100	XXX	000	10	1	0	0	0	1
	xx xxx	x		0010	XXX	000	10	0	0	0	1	1
	xx xxx	x		0010	XXX	000	10	0	0	0	1	1

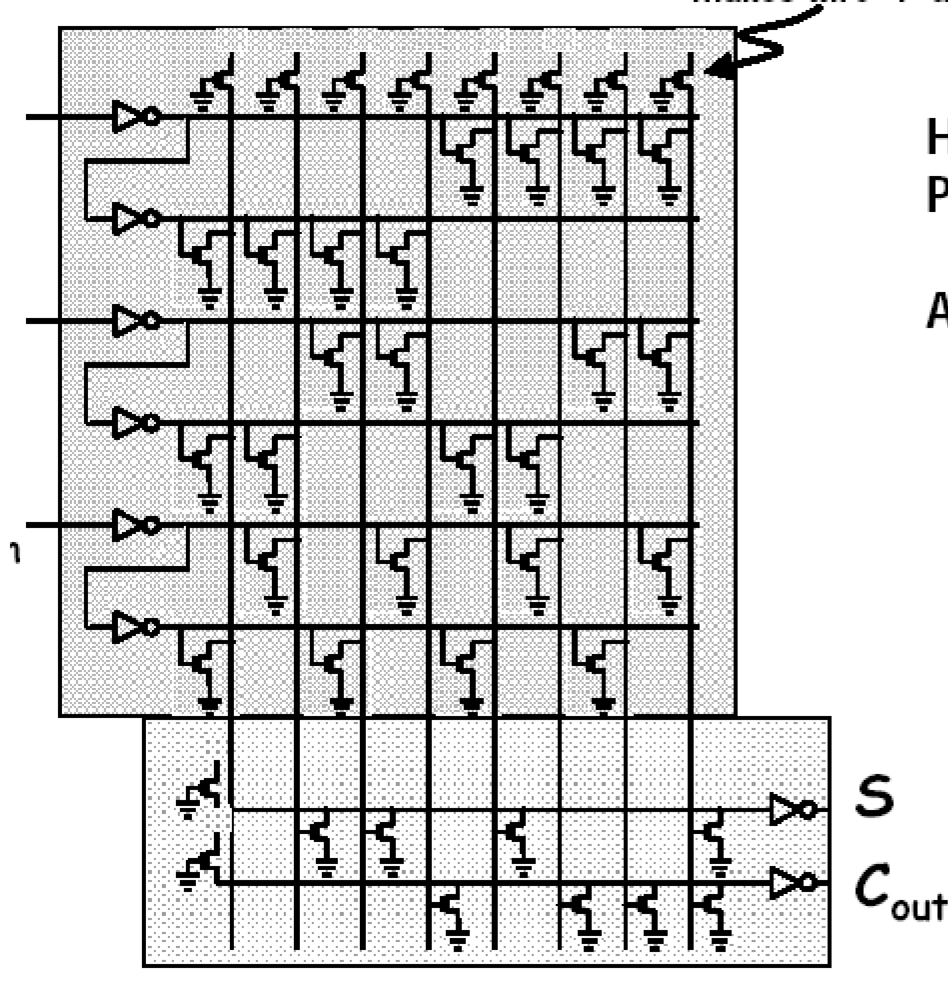
it ROM address

64 ROM Addresses with identical content

18-bit ROM outputs

ROM Implementation Technology

PFET with gate tied to ground = resistor pullup that makes wire "1" unless one of the NFET pulldowns is on.

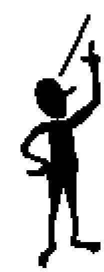


Hardwired AND logic
Programmable OR logic

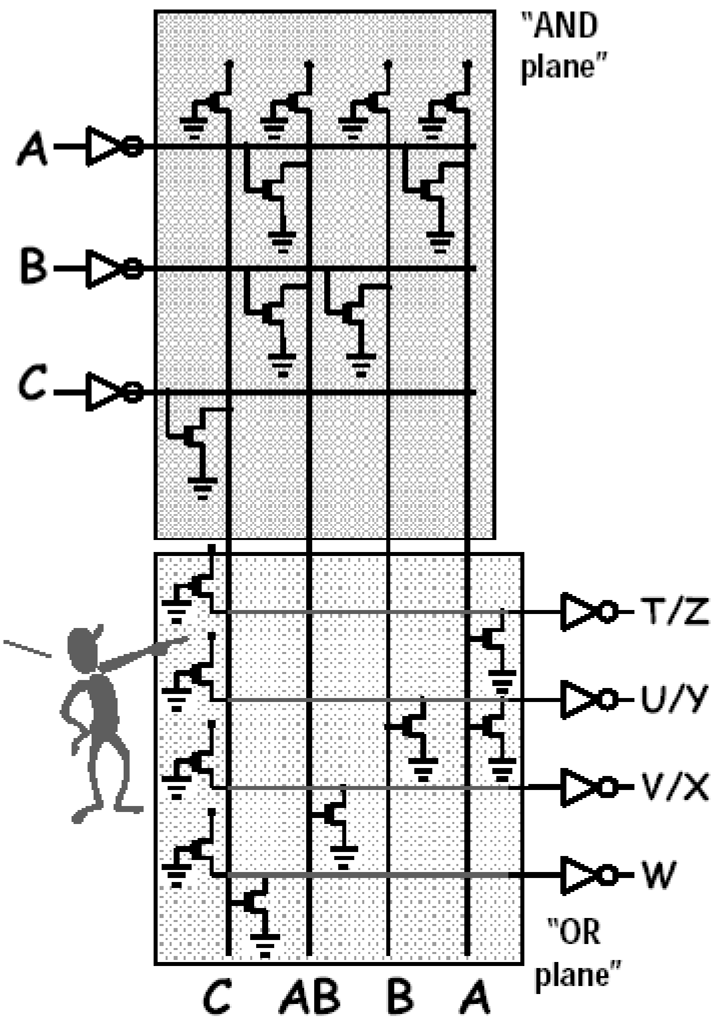
Advantages:
- Very regular design (can be entirely automated)

Problems:
- Active Pull-ups (Static Power)
- Long metal runs (Large Caps)
- Slow

JARGON: Inputs to a ROM are called ADDRESSES. The decoder's outputs are called WORD LINES, and the outputs lines of the selector are called BIT LINES.



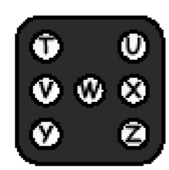
PLA 7-sided Die implementation



PLAs like ROMs support the synthesis of arbitrary logic functions using SOP implementations.

However, they allow for
- minimal realizations
- smaller (faster) arrays

Regular structure
- automatic generation
- easy design
- still slower than optimized gates



Building the Easy-I C-Unit
2 Approaches

Harwired

– Apply well known sequential circuit techniques

Micro-programmed

– Treat state transition table as a program

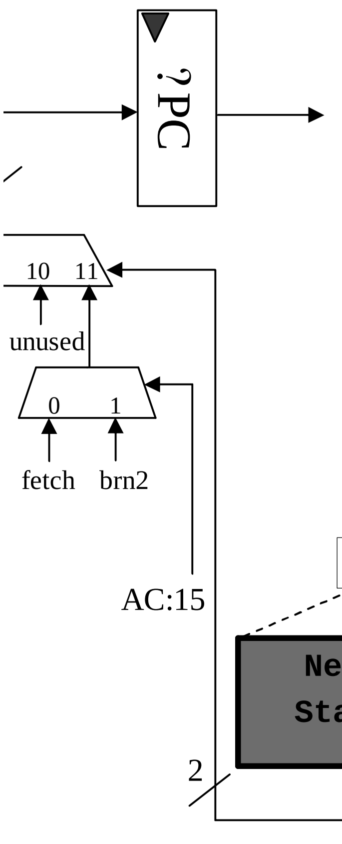
A
? prog

– Build a new abstraction layer

The Microprogramming abstraction level

Building the Easy-I C-Unit

Micro-programmed Approach



reset1	reset2	01	xx	00	01	X	0	0	X	0	X
reset2	fetch	01	xx	00	10	1	0	0	0	1	X
fetch	xxxx	00	xx	00	11	X	1	0	X	0	X
store1	store2	01	xx	00	11	X	0	0	1	1	X
store2	fetch	01	xx	10	10	1	0	0	0	1	1
load1	load2	01	xx	00	11	X	0	0	1	1	X
load2	load3	01	xx	01	11	X	1	0	X	0	X
load3	fetch	01	xx	01	11	X	1	0	X	0	X
brn1	xxxx	11	xx	00	10	1	0	0	0	1	X
brn2	fetch	01	xx	00	10	0	0	0	1	1	X
jump	fetch	01	xx	00	10	0	0	0	1	1	X

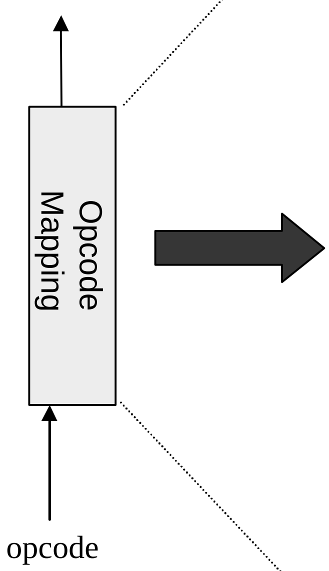
?Prog
(Comb

Next State	Bra nch	ALU op	Mem OP	PC sel	PC is	DI le	AC le	AO sel	AO le	EDB sel

DataPath Control

Finding the first execute state

Combinational Logic



Instruction	Opcode	AC:15	First Execute State
Comp	00 000	x	sopr
ShR	00 001	x	sopr
BrN	00 010	1	brn1
Jump	00 011	x	jump
Store	00 100	x	store1
Load	00 101	x	load1
And	00 110	x	aopr
Add	00 111	x	aopr

Summary

What we know?

From

Instruction Set Architecture

To

Processor Implementation

What Next?

How do we get here In the first place?

Instruction Design