50.002 Computation Structures Building the Beta & Exceptions

Oliver Weeger

2018 Term 3, Week 9, Session 2

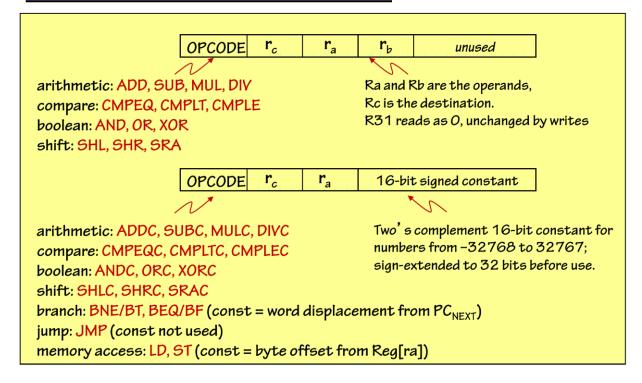




Building the Beta: Implementing the data paths for the ISA



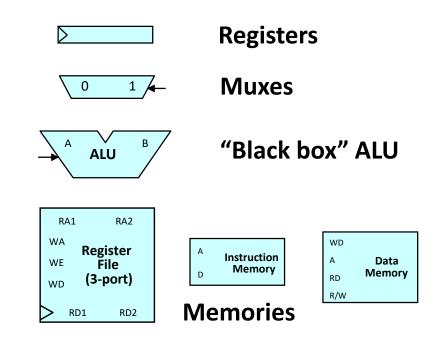
Instruction Set Architecture:



Classes of instructions:

- 1. ALU: OP & OPC
- 2. LD & ST
- 3. Branches
- 4. Exceptions

Digital Circuit Components:



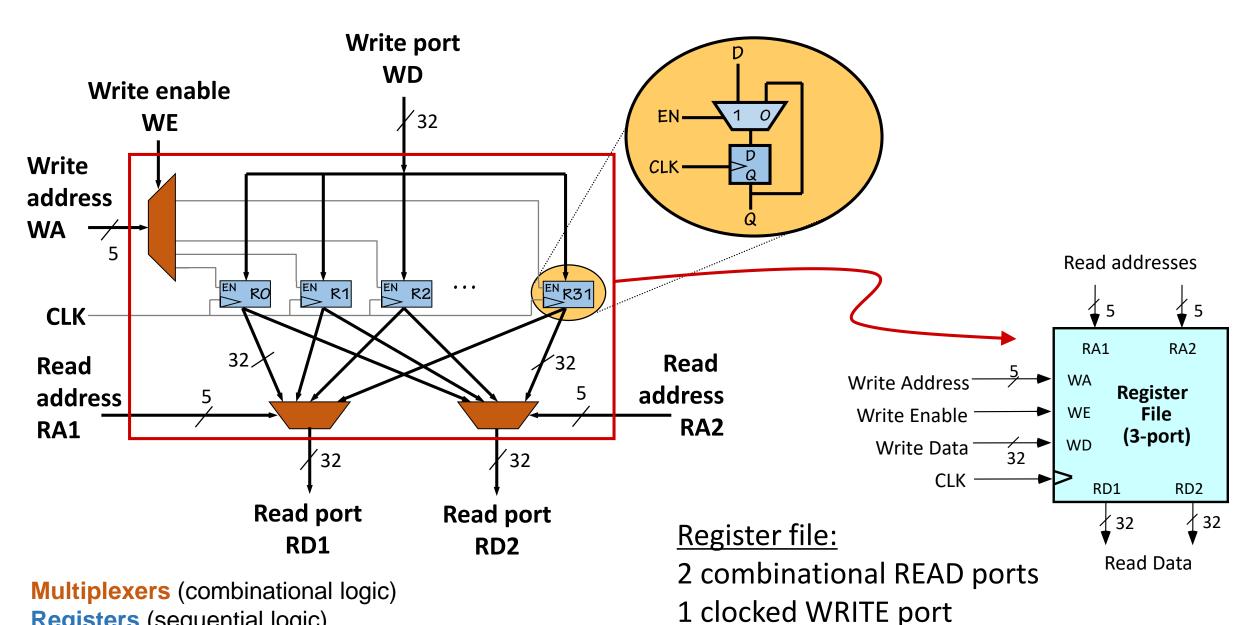


Hardware implementation of registers, PC, control logic & data paths

Register file

Registers (sequential logic)

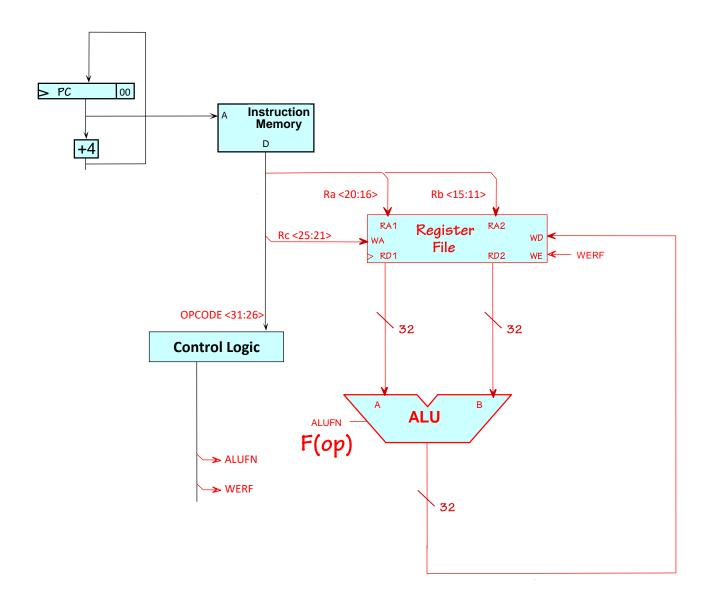




Slide 3

Data path for ALU OP instruction





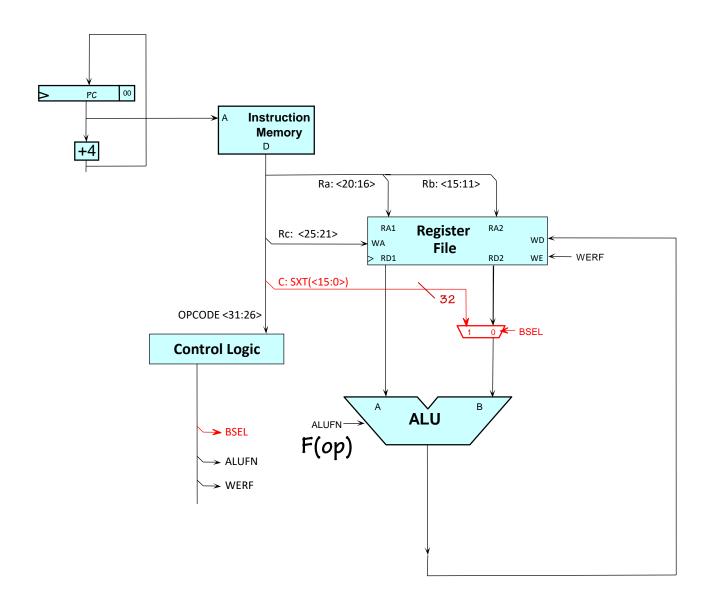
	OP	OPC	CD	ST	JMP	ВЕО	BNE	LDR	lllop	IRQ
ALUFN	F(op)									
WERF	1									

10 X X X X Rc Ra Rb (UNUSED)

Operate class: $Reg[Rc] \leftarrow Reg[Ra]$ op Reg[Rb]

Data path for ALU OPC instruction





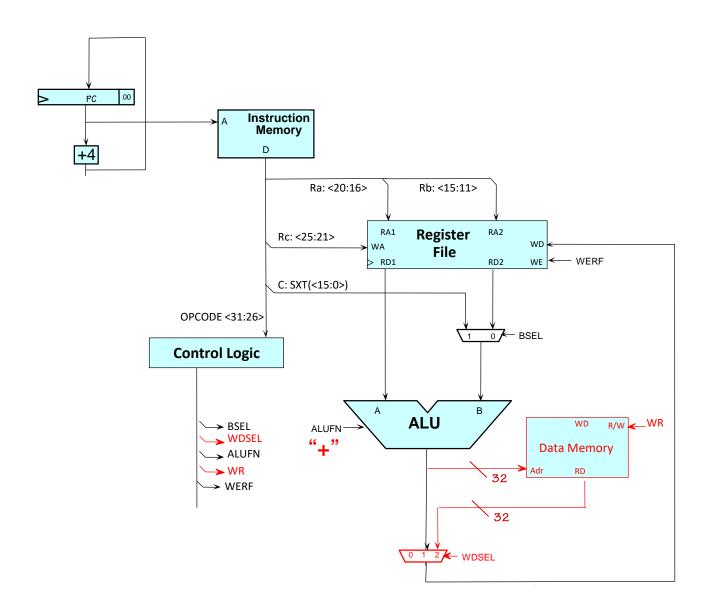
	OP	OPC	CD	ST	JMP	ВЕО	BNE	LDR	lllop	IRQ
ALUFN	F(op)	F(op)								
WERF	1	1								
BSEL	0	1								

11XXXX Rc Ra Literal C (signed)

Operate class: $Reg[Rc] \leftarrow Reg[Ra]$ op SXT(C)

Data path for LD instruction





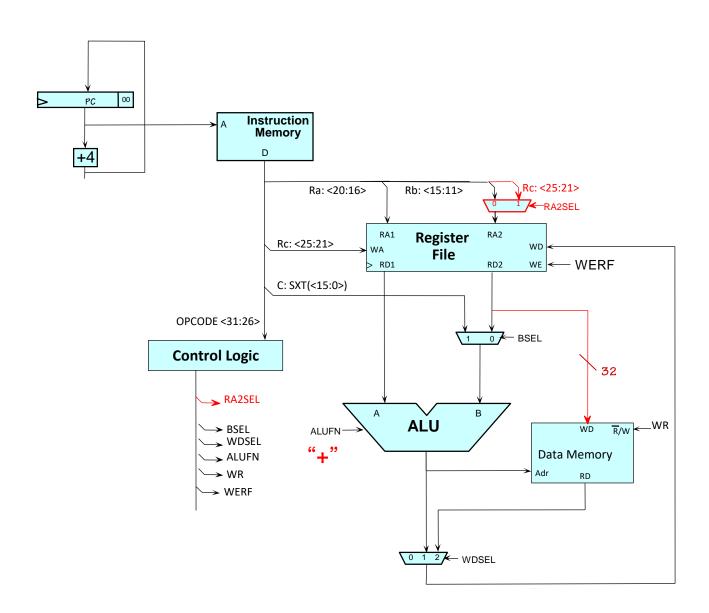
	OP	OPC	П	ST	JMP	ВЕО	BNE	LDR	Illop	IRQ
ALUFN	F(op)	F(op)	"+"							
WERF	1	1	1							
BSEL	0	1	1							
WDSEL	1	1	2							
WR	0	0	0							

011000 Rc Ra Literal C (signed)

LD: $Reg[Rc] \leftarrow Mem[Reg[Ra] + SXT(C)]$

Data path for ST instruction





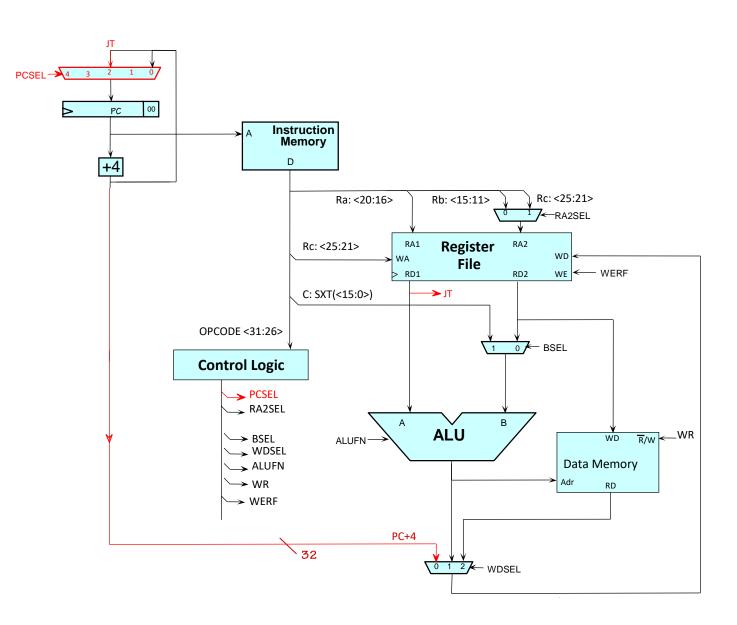
	OP	OPC	П	ST	JMP	ВЕQ	BNE	LDR	Illop	IRQ
ALUFN	F(op)	F(op)	"+"	"+"						
WERF	1	1	1	0						
BSEL	0	1	1	1						
WDSEL	1	1	2							
WR	0	0	0	1						
RA2SEL	0			1						

011001 Rc Ra Literal C (signed)

ST: $Mem[Reg[Ra]+SXT(C)] \leftarrow Reg[Rc]$

Data path for JMP instruction





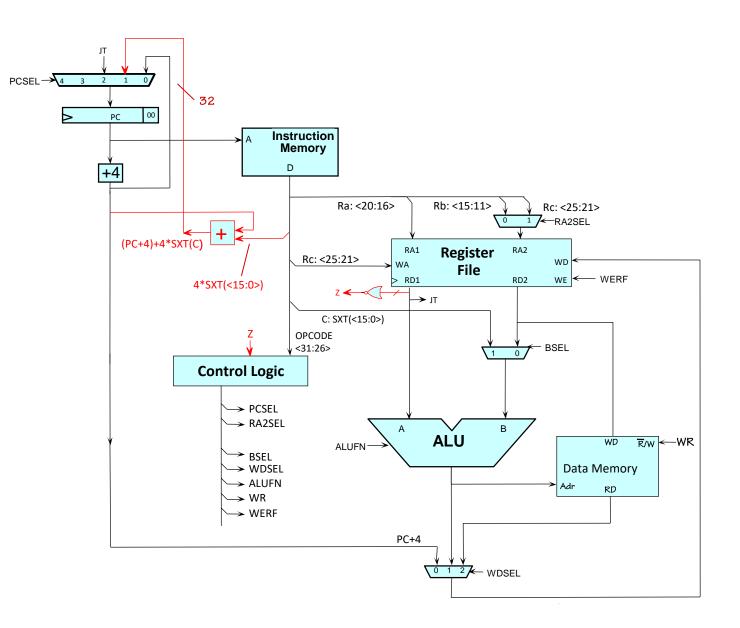
	OP	OPC	9	ST	JMP	ВЕQ	BNE	LDR	lllop	IRQ
ALUFN	F(op)	F(op)	"+"	"+"						
WERF	1	1	1	0	1					
BSEL	0	1	1	1						
WDSEL	1	1	2		0					
WR	0	0	0	1	0					
RA2SEL	0			1						
PCSEL	0	0	0	0	2					

O11011 Rc Ra Literal C (signed)

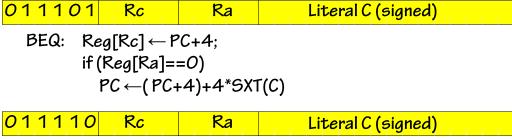
JMP: $Reg[Rc] \leftarrow PC+4$; $PC \leftarrow Reg[Ra]$

Data path for BEQ & BNE instruction





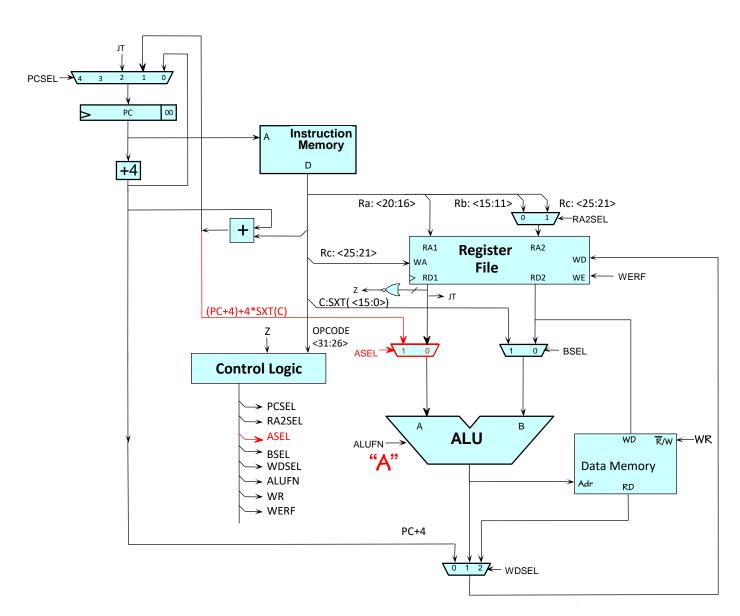
	OP	OPC	П	ST	JMP	ВЕО	BNE	LDR	lllop	IRQ
ALUFN	F(op)	F(op)	"+"	"+"						
WERF	1	1	1	0	1	1	1			
BSEL	0	1	1	1						
WDSEL	1	1	2		0	0	0			
WR	0	0	0	1	0	0	0			
RA2SEL	0			1						
PCSEL	0	0	0	0	2	Z?1:0	Z?0:1			



BNE:
$$Reg[Rc] \leftarrow PC+4$$
;
if $(Reg[Ra] \neq 0)$
 $PC \leftarrow (PC+4)+4*SXT(C)$

Data path for LDR instruction





	OP	OPC	П	ST	JMP	ВЕО	BNE	LDR	Illop	IRQ
ALUFN	F(op)	F(op)	"+"	"+"				"A"		
WERF	1	1	1	0	1	1	1	1		
BSEL	0	1	1	1						
WDSEL	1	1	2		0	0	0	2		
WR	0	0	0	1	0	0	0	0		
RA2SEL	0			1						
PCSEL	0	0	0	0	2	Z?1:0	Z?0:1	0		
ASEL	0	0	0	0				1		

O 1 1 1 1 1 Rc Ra Literal C (signed)

LDR: $Reg[Rc] \leftarrow Mem[(PC + 4) + 4*SXT(C)]$

Used to reference addresses & other large constants

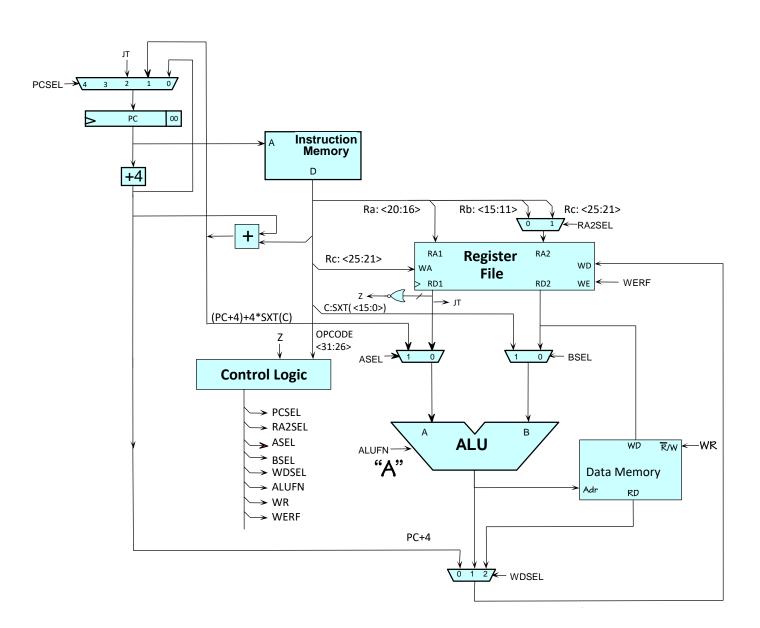
C: X = X * 123456;

BETA:

LD(X, r0)
LDR(c1, r1)
MUL(r0, r1, r0)
ST(r0, X)
...
c1: LONG(123456)

The Beta CPU data path (so far)

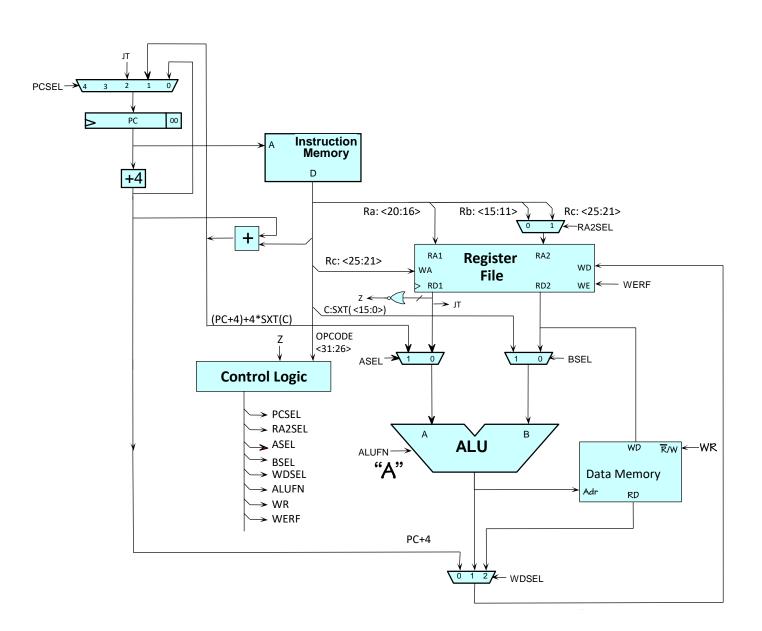




	OP	OPC	ΓD	ST	JMP	ВЕQ	BNE	LDR	Illop	IRQ
ALUFN	F(op)	F(op)	"+"	"+"				"A"		
WERF	1	1	1	0	1	1	1	1		
BSEL	0	1	1	1						
WDSEL	1	1	2		0	0	0	2		
WR	0	0	0	1	0	0	0	0		
RA2SEL	0			1						
PCSEL	0	0	0	0	2	Z?1:0	Z?0:1	0		
ASEL	0	0	0	0				1		

Exercise: extending the Beta (1)





	OP	OPC	CD	ST	JMP	BEQ	BNE	LDR	TDX
ALUFN	F(op)	F(op)	"+"	"+"				"A"	"+"
WERF	1	1	1	0	1	1	1	1	1
BSEL	0	1	1	1					0
WDSEL	1	1	2		0	0	0	2	2
WR	0	0	0	1	0	0	0	0	0
RA2SEL	0			1					0
PCSEL	0	0	0	0	2	Z?1:0	Z?0:1	0	0
ASEL	0	0	0	0				1	0

How could we add an instruction

LDX(R0,R1,R2)

as a short-cut for

ADD(R1,R0,R0)

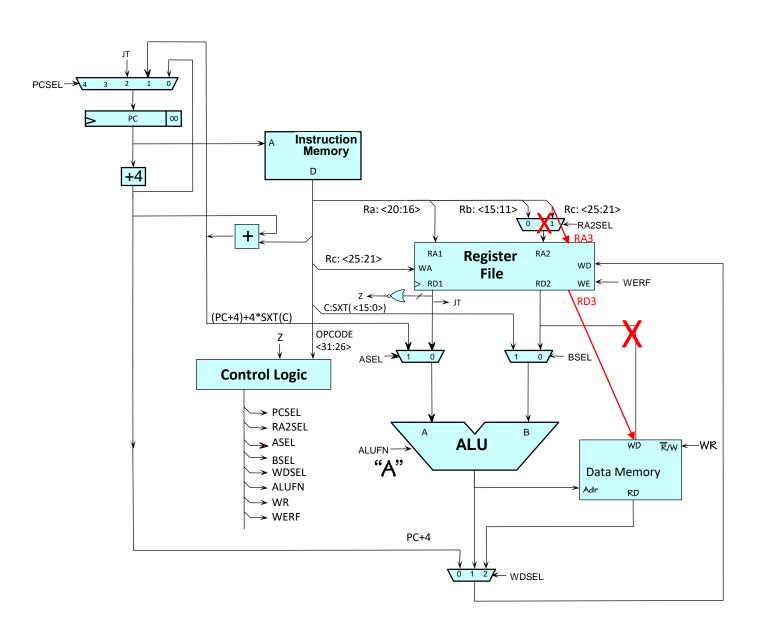
LD(R0,0,R2) ?

Register-transfer language expression:
Reg[Rc] ← Mem[Reg[Ra] + Reg[Rb]]

LDX (Ra, Rb, Rc)

Exercise: extending the Beta (2)





	OP	OPC	ΠD	ST	JMP	BEQ	BNE	LDR	STX	
ALUFN	F(op)	F(op)	"+"	"+"				"A"	" + "	
WERF	1	1	1	0	1	1	1	1	0	
BSEL	0	1	1	1					0	
WDSEL	1	1	2		0	0	0	2	0	
WR	0	0	0	1	0	0	0	0	1	
RA2SEL	0			1					0	
PCSEL	0	0	0	0	2	Z?1:0	Z?0:1	0	0	
ASEL	0	0	0	0				1	0	

How could we add an instruction

STX(R2,R0,R1)

as a short-cut for

ADD(R1,R0,R0) ST(R2,0,R0) ?

Register-transfer language expression: Mem[Reg[Ra] + Reg[Rb]] \leftarrow Reg[Rc] STX (Rc, Rb, Ra)

Must amend data path & register file!
Register file needs another RA/RD port!
Could eliminate RA2SEL mux!

Exceptions & Interrupts



- Exceptions (synchronous/software interrupts): Something bad happens ...
 - Illegal OPCODE in instruction word
 - Reference to non-existent memory
 - Divide by 0
- Asynchronous/hardware interrupts: Something unexpected, but critical happens ...
 - User hits a key
 - A packet comes via the network
 - In general: input from I/O hardware device

Interrupt handling in software:

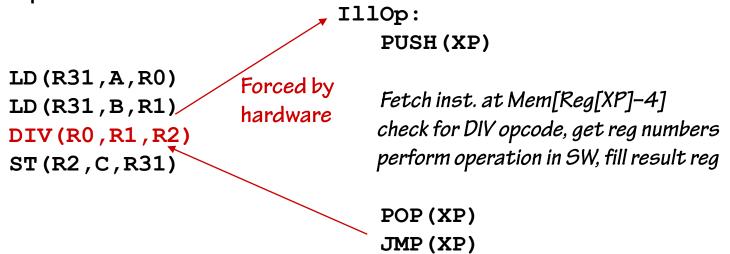
- Interrupt running program/task
- Invoke exception handler (a procedure call)
- Return to continue execution
- Transparency to interrupted program (i.e., the interrupt handler should preserve the current state of the machine, not change the memory contents)

Implementation of interrupts/exceptions



- Exception handling:
 - Do not execute current instruction
 - Instead fake a "forced" procedure call
 - Save current PC (actually PC + 4) \rightarrow Exception pointer register XP=R30
 - Load PC with exception pointer
 (0x4 for synch. exception, 0x8 for asynch. exceptions)

Example: DIV unimplemented



R27=BP

R28=LP

R29=SP

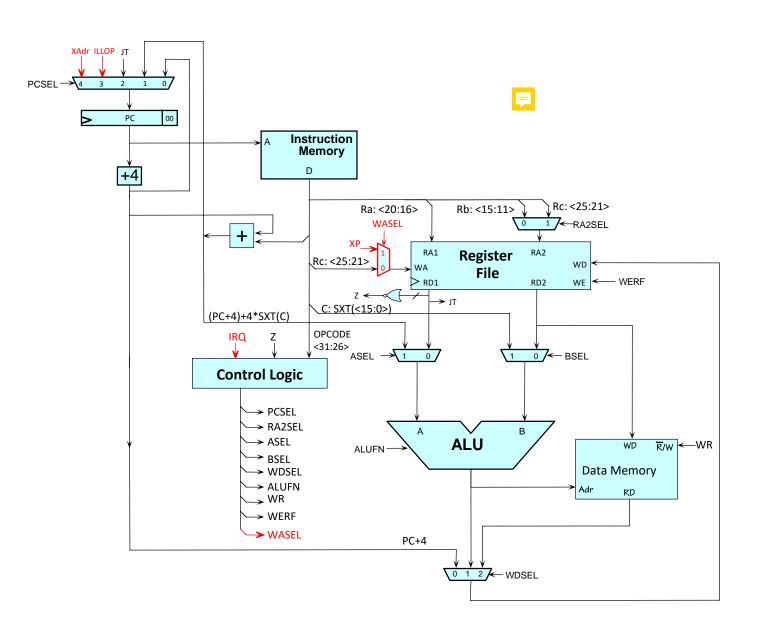
R30=XP

R31 = 0

R0: return value

Exceptions in the Beta CPU data path





	OP	OPC	П	ST	JMP	ВЕQ	BNE	LDR	lllop	IRQ
ALUFN	F(op)	F(op)	"+"	"+"				"A"		
WERF	1	1	1	0	1	1	1	1	1	1
BSEL	0	1	1	1						
WDSEL	1	1	2		0	0	0	2	0	0
WR	0	0	0	1	0	0	0	0	0	0
RA2SEL	0			1						
PCSEL	0	0	0	0	2	Z?1:0	Z?0:1	0	3	4
ASEL	0	0	0	0				1		
WASEL	0	0	0		0	0	0	0	1	1

Bad Opcode:

Reg[XP] \leftarrow PC+4 PC \leftarrow IIIOp

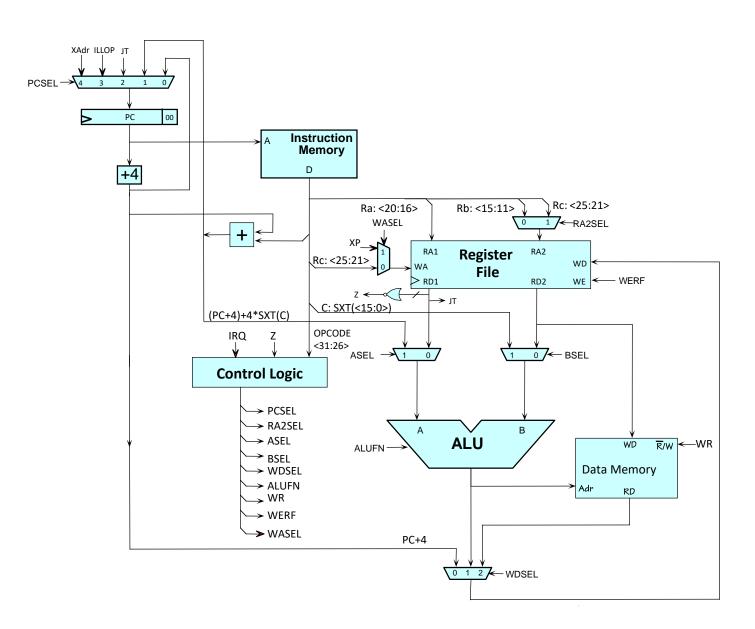
Other exceptions:

Reg[XP] \leftarrow PC+4 PC \leftarrow Xadr



Finally: The Beta CPU data path & control logic





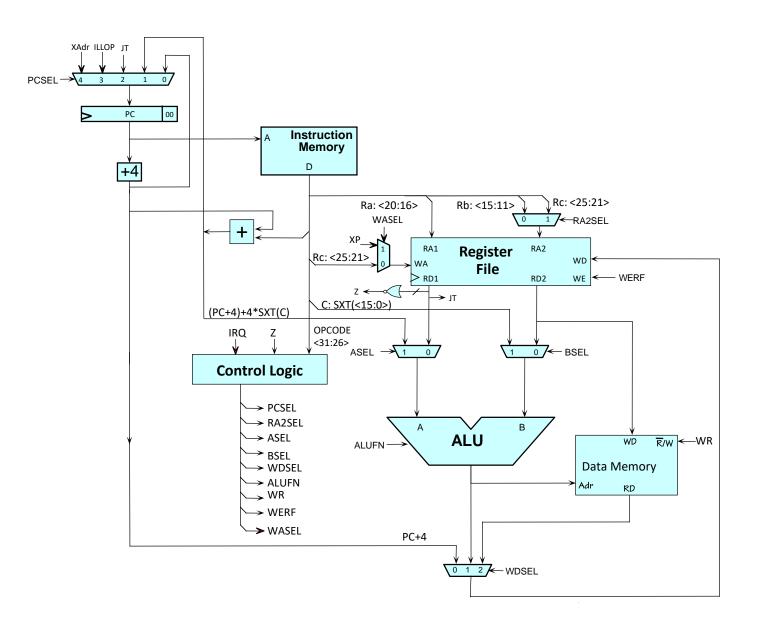
	OP	OPC		ST	JMP	ВЕQ	BNE	LDR	lllop	IRQ
ALUFN	F(op)	F(op)	"+"	"+"				"A"		
WERF	1	1	1	0	1	1	1	1	1	1
BSEL	0	1	1	1						
WDSEL	1	1	2		0	0	0	2	0	0
WR	0	0	0	1	0	0	0	0	0	0
RA2SEL	0			1						
PCSEL	0	0	0	0	2	Z?1:0	Z?0:1	0	3	4
ASEL	0	0	0	0				1		
WASEL	0	0	0		0	0	0	0	1	1

Control logic implementation:

- ROM indexed by OPCODE, external branch & trap logic
- Programmable logic array (PLA)
- "Random" logic (e.g., standard cell gates)

Exercise: Beta dynamic discipline





	OP	OPC	9	ST	JMP	ВЕД	BNE	LDR	Illop	IRQ
ALUFN	F(op)	F(op)	"+"	"+"				"A"		
WERF	1	1	1	0	1	1	1	1	1	1
BSEL	0	1	1	1						
WDSEL	1	1	2		0	0	0	2	0	0
WR	0	0	0	1	0	0	0	0	0	0
RA2SEL	0			1						
PCSEL	0	0	0	0	2	Z?1:0	Z?0:1	0	3	4
ASEL	0	0	0	0				1		
WASEL	0	0	0		0	0	0	0	1	1

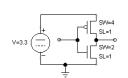
Q: Which control signal should become valid first to to ensure the smallest possible clock period for the Beta?

A: RA2SEL

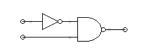


Where are we? The 50.002 roadmap

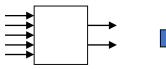


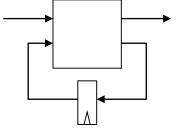








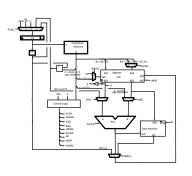




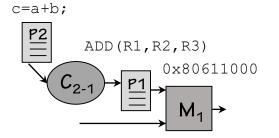
Digital abstraction, Fets & CMOS, Static discipline Logic gates &
Boolean Algebra
(AND, OR, NAND,
NOR, etc.)

Combinational logic circuits:
Truth tables,
Multiplexers, ROMs

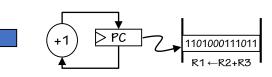
Sequential logic &
Finite State Machines:
Dynamic Discipline,
Registers, State
Transition Diagrams



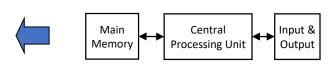
β CPU Architecture



Software Abstraction, Assembler Language, C Compilation



Instruction Set
Architecture: β RISC,
ALU (lab)



Programmability & von Neumann model, General-Purpose Computer



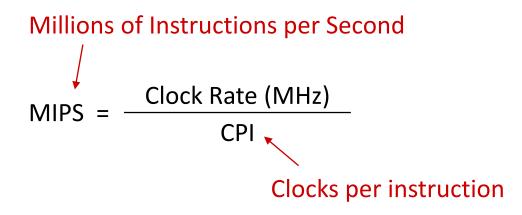
CPU design trade-offs

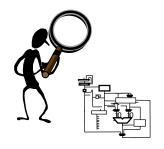




Maximum Performance:

Measured by the number of instructions executed per second.





Minimum Cost:

Measured by the size of the circuit.



Best Performance/Price:

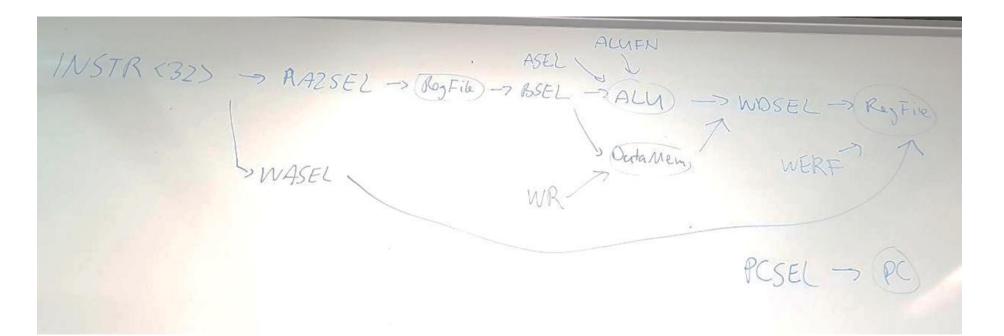
Measured by the ratio of MIPS to size. In power-sensitive applications MIPS/Watt is important, too.

Exercise: CPU design trade-offs



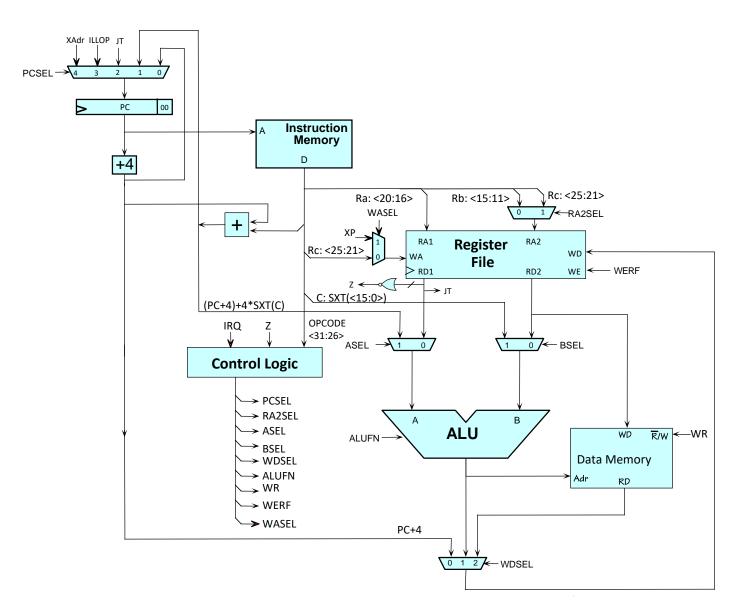
Which machine would you choose?

Model Clock rate CPI		CPI	Benchmark test: # instr. exec.	MIPS	Benchmark runtime			
(a)	40 MHz	2.0	3,600,000	20.0	0.18 s → fastest			
(b)	100 MHz	10.0	1,900,000	10.0	0.19 s			
(c)	60 MHz	3.0	4,200,000	20.0	0.21 s			



Summary: The Beta CPU data path & control logic





								HNOLOGY AND DESIGN			
	dО	OPC	П	ST	JMP	BEQ	BNE	LDR	Illop	IRQ	
ALUFN	F(op)	F(op)	"+"	"+"				"A"		!	
WERF	1	1	1	0	1	1	1	1	1	1	
BSEL	0	1	1	1							
WDSEL	1	1	2		0	0	0	2	0	0	
WR	0	0	0	1	0	0	0	0	0	0	
RA2SEL	0			1							
PCSEL	0	0	0	0	2	Z?1:0	Z?0:1	0	3	4	
ASEL	0	0	0	0				1			
WASEL	0	0	0		0	0	0	0	1	1	

Summary:

- We have used digital circuit components for the hardware implementation of the β CPU:
 - ✓ ALU, PC
 - ✓ Register File
 - ✓ Control Logic
 - ✓ Data paths
- Now we can write (assembly & \mathbb{C}) programs for the β and execute them!
- To follow: Memory, VMs, OS