CS 110
Computer Architecture
Single-Cycle CPU
Datapath & Control

Instructor:
Sören Schwertfeger

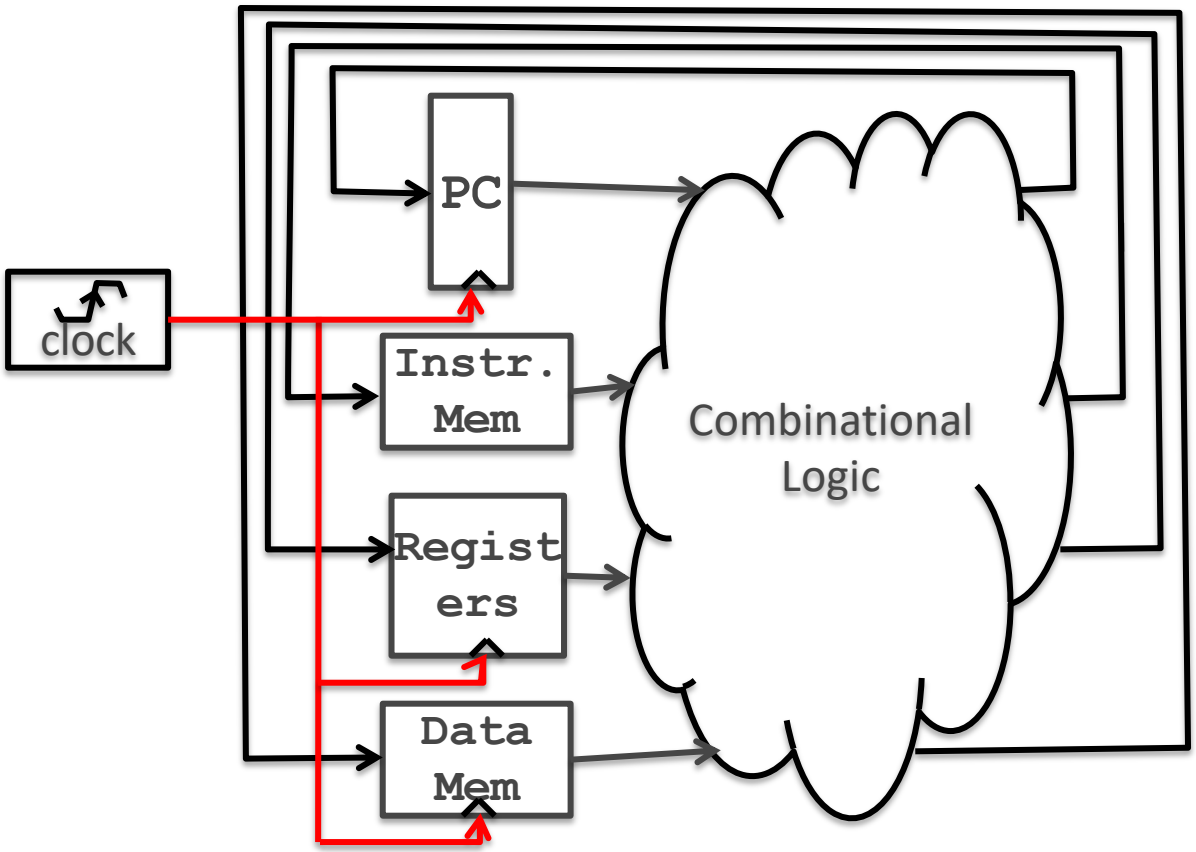
http://shtech.org/courses/ca/

School of Information Science and Technology SIST

ShanghaiTech University

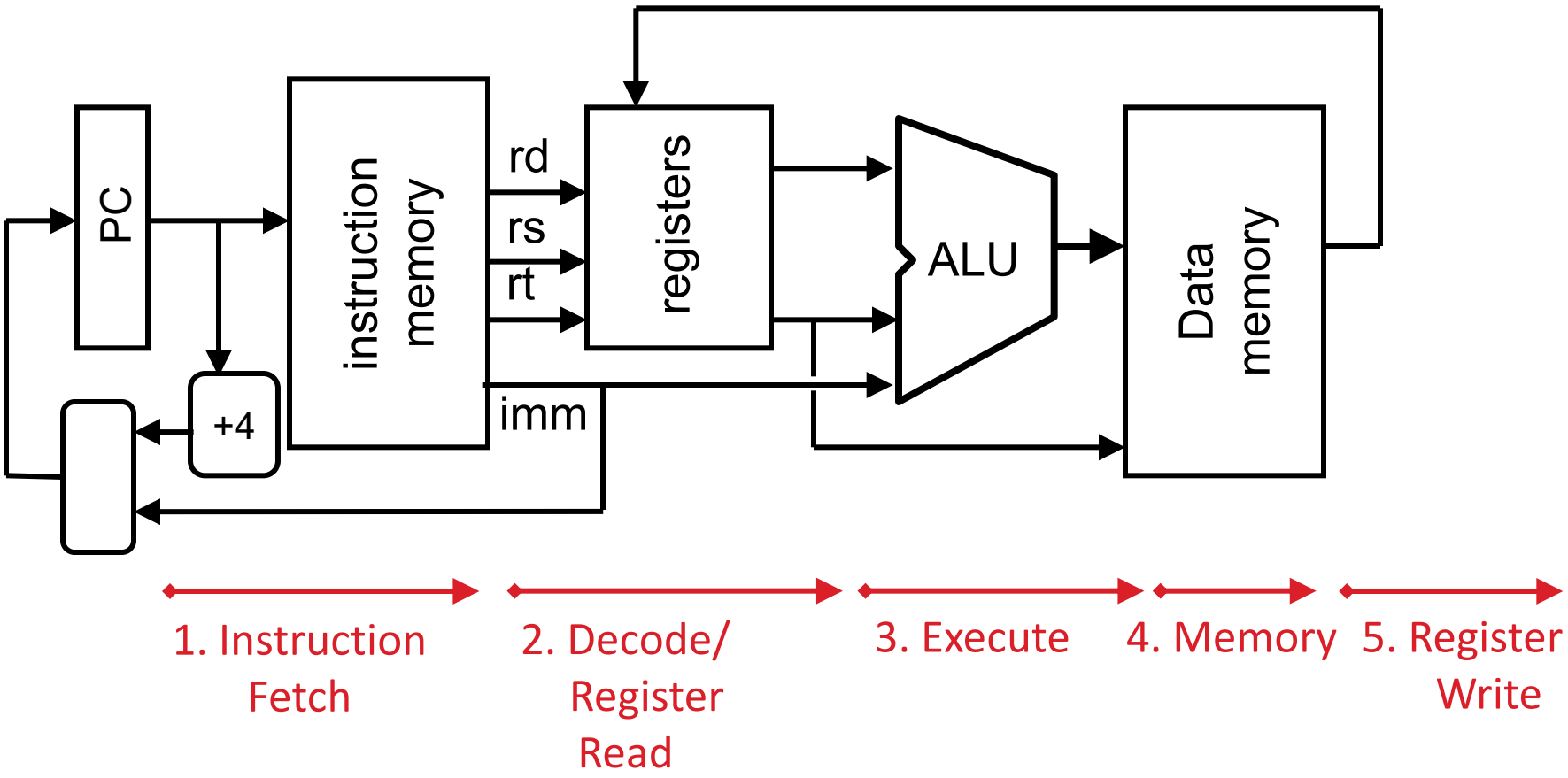
Slides based on UC Berkley's CS61C

One-Instruction-Per-Cycle RISC-V Machine



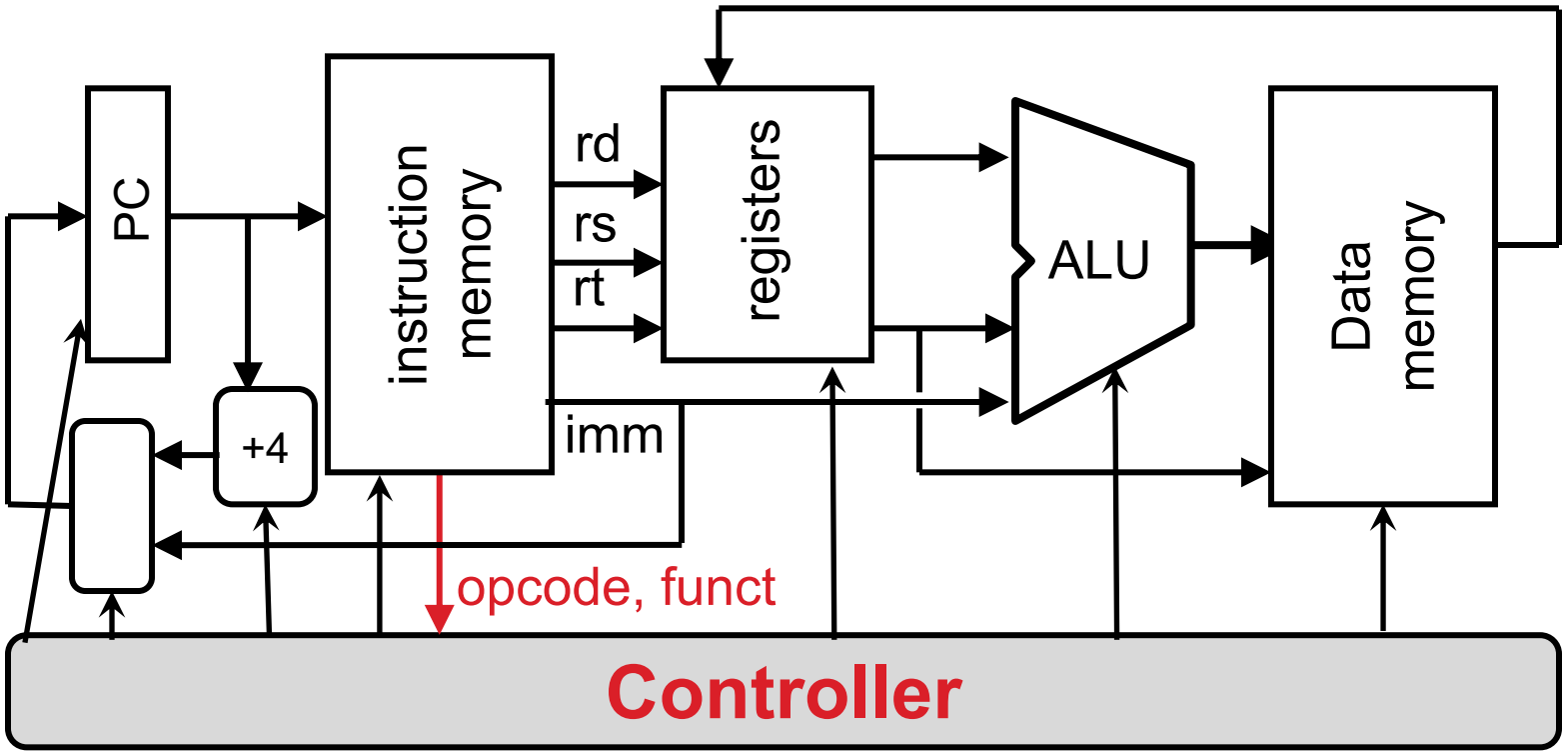
- One clock tick => one instruction
- Current state outputs => inputs to combinational logic => outputs settle at the values of state before next clock edge
- Rising clock edge:
  - all state elements are updated with combinational logic outputs
  - execution moves to next clock cycle

Stages of Execution on Datapath



Datapath and Control

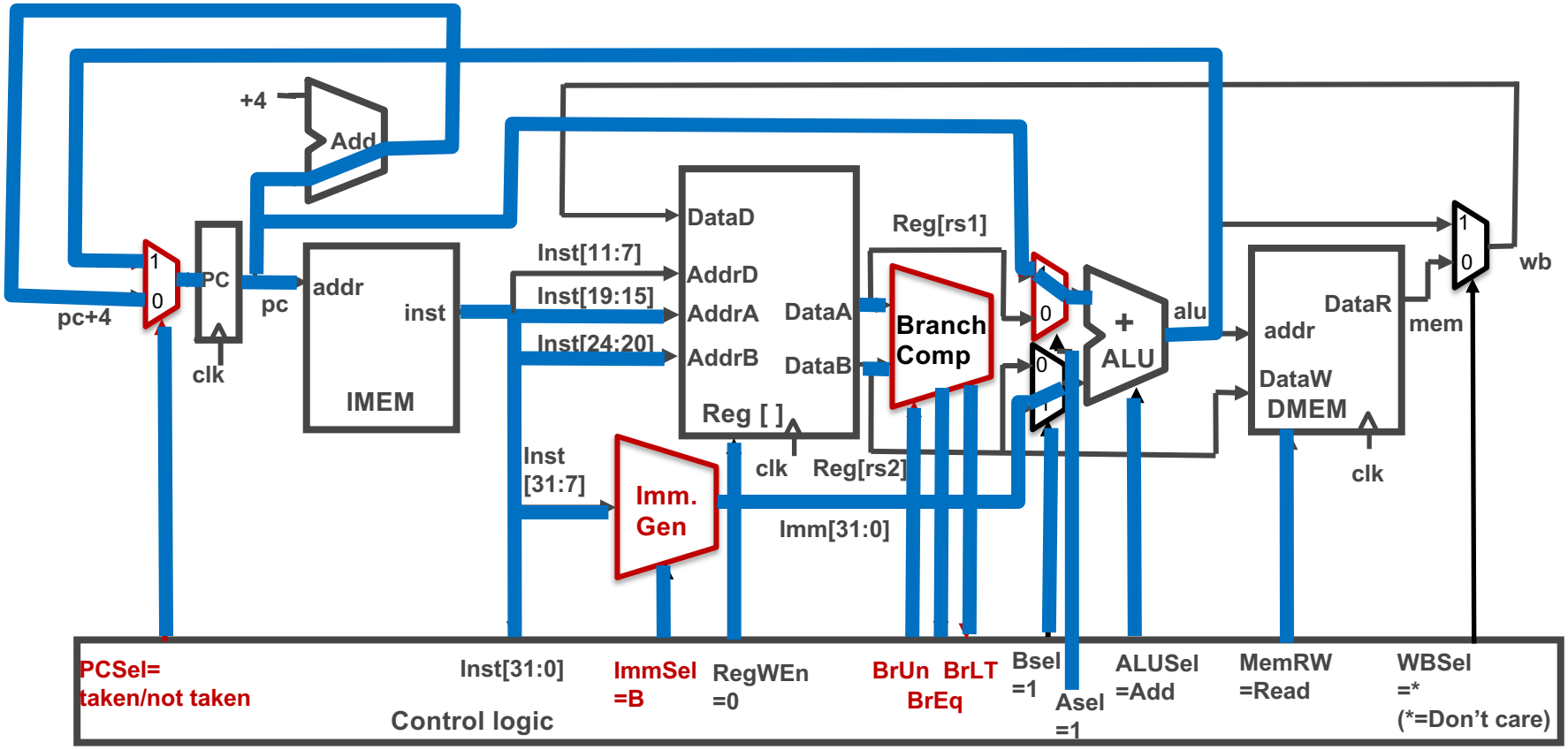
- Datapath designed to support data transfers required by instructions
- Controller causes correct transfers to happen



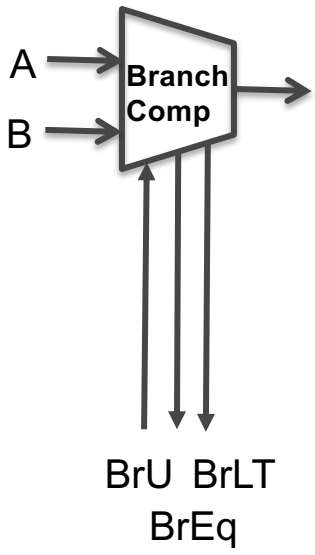
Branches

- Different change to the state:
  - $\texttt{PC} = \begin{cases} \texttt{PC + 4}, & \texttt{branch not taken} \\ \texttt{PC + immediate}, & \texttt{branch taken} \end{cases}$
- Six branch instructions:
  `BEQ, BNE, BLT, BGE, BLTU, BGEU`
- Need to compute `PC + immediate` and to compare values of `rs1` and `rs2`
  - But have only one ALU – need more hardware

Adding Branches



Branch Comparator

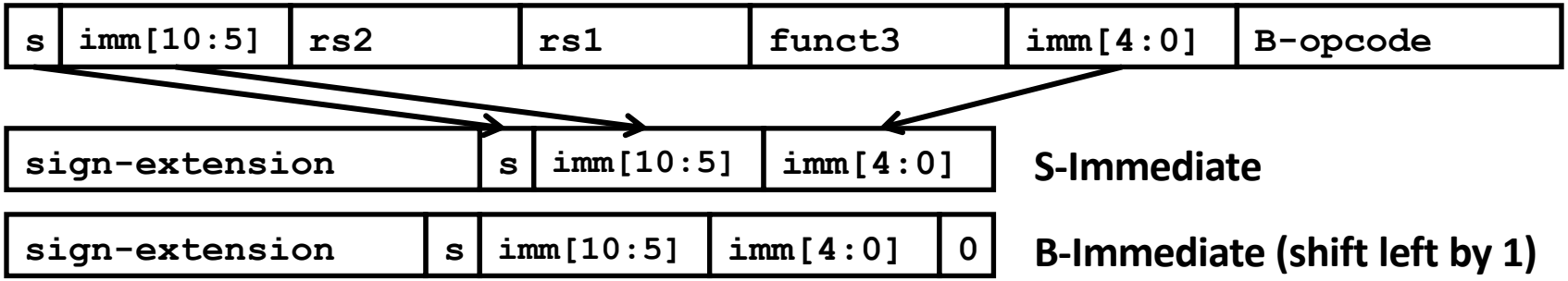


- BrEq = 1, if A=B
- BrLT = 1, if A < B
- BrUn =1 selects unsigned comparison for BrLT, 0=signed
- BGE branch: A >= B, if $\overline{A<B}$

$$\overline{A<B} = !(A<B)$$

Branch Immediates (In Other ISAs)

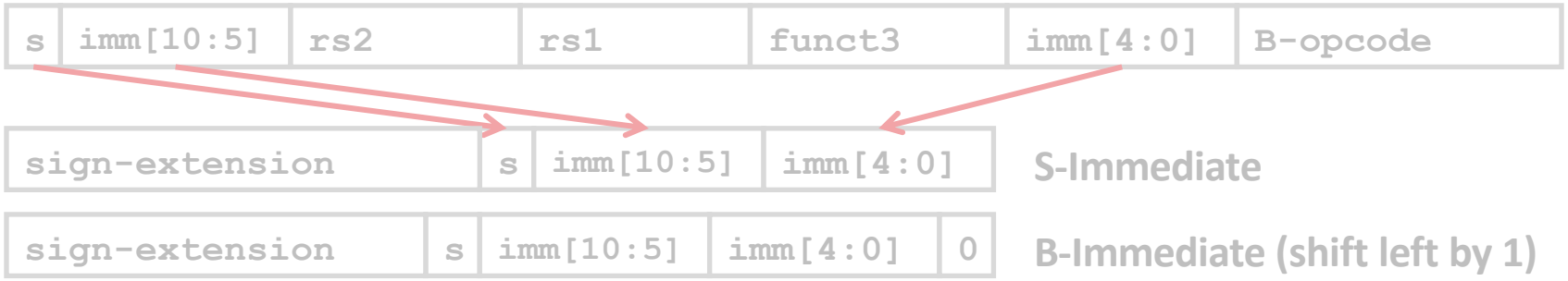
- 12-bit immediate encodes PC-relative offset of -4096 to +4094 bytes in multiples of 2 bytes
- Standard approach: Treat immediate as in range -2048..+2047, then shift left by 1 bit to multiply by 2 for branches



Each instruction immediate bit can appear in one of two places in output immediate value – so need one 2-way mux per bit

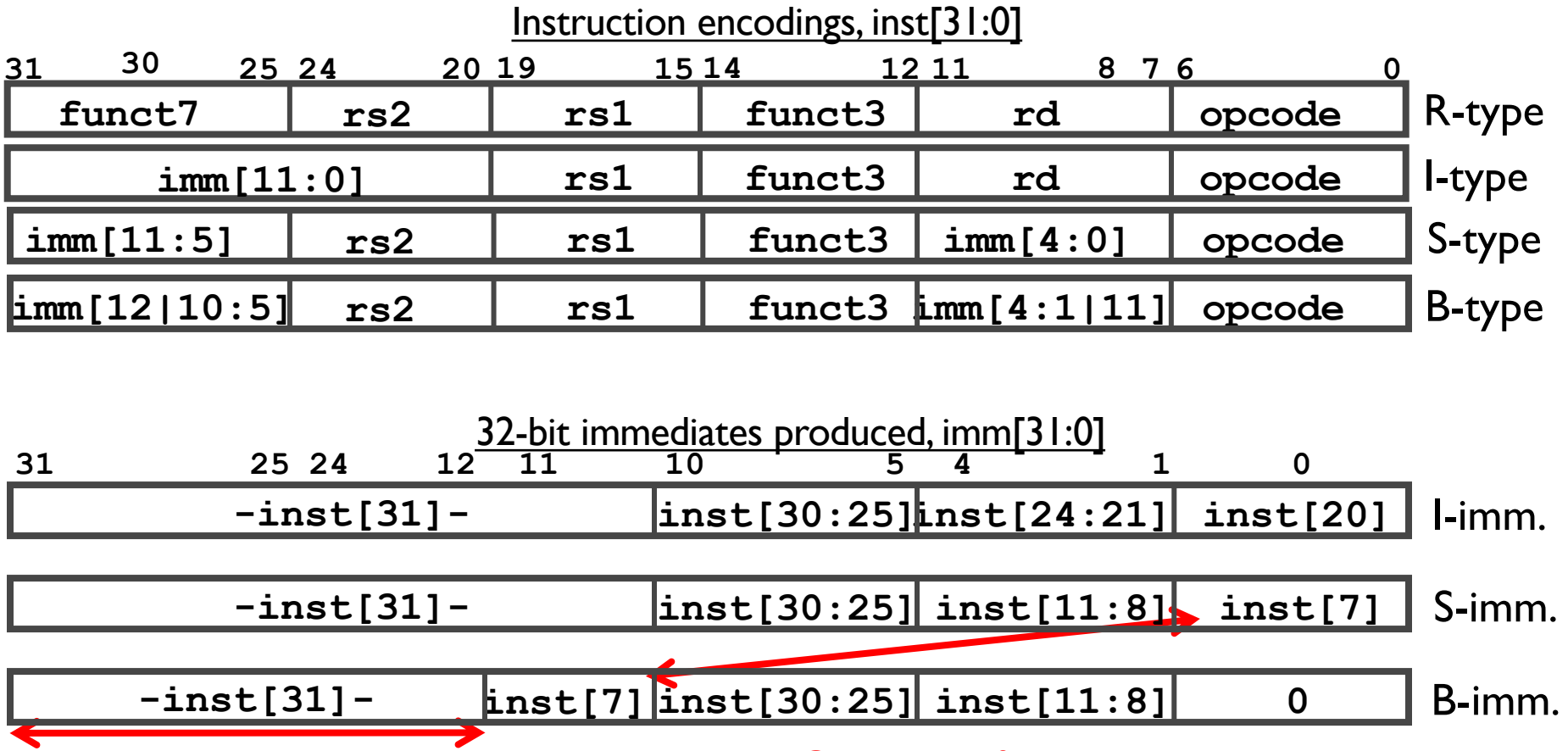
RISC-V Branch Immediates

- 12-bit immediate encodes PC-relative offset of -4096 to +4094 bytes in multiples of 2 bytes
- RISC-V approach: keep 11 immediate bits in fixed position in output value, and rotate LSB of S-format to be bit 12 of B-format

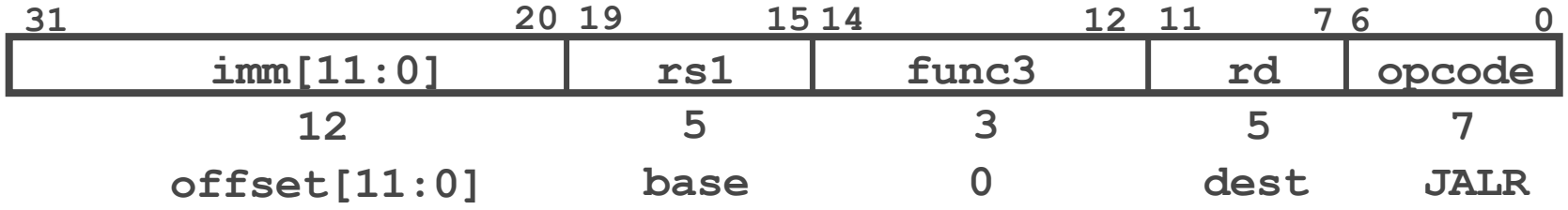


Only one bit changes position between S and B, so only need a single-bit 2-way mux:

RISC-V Immediate Encoding

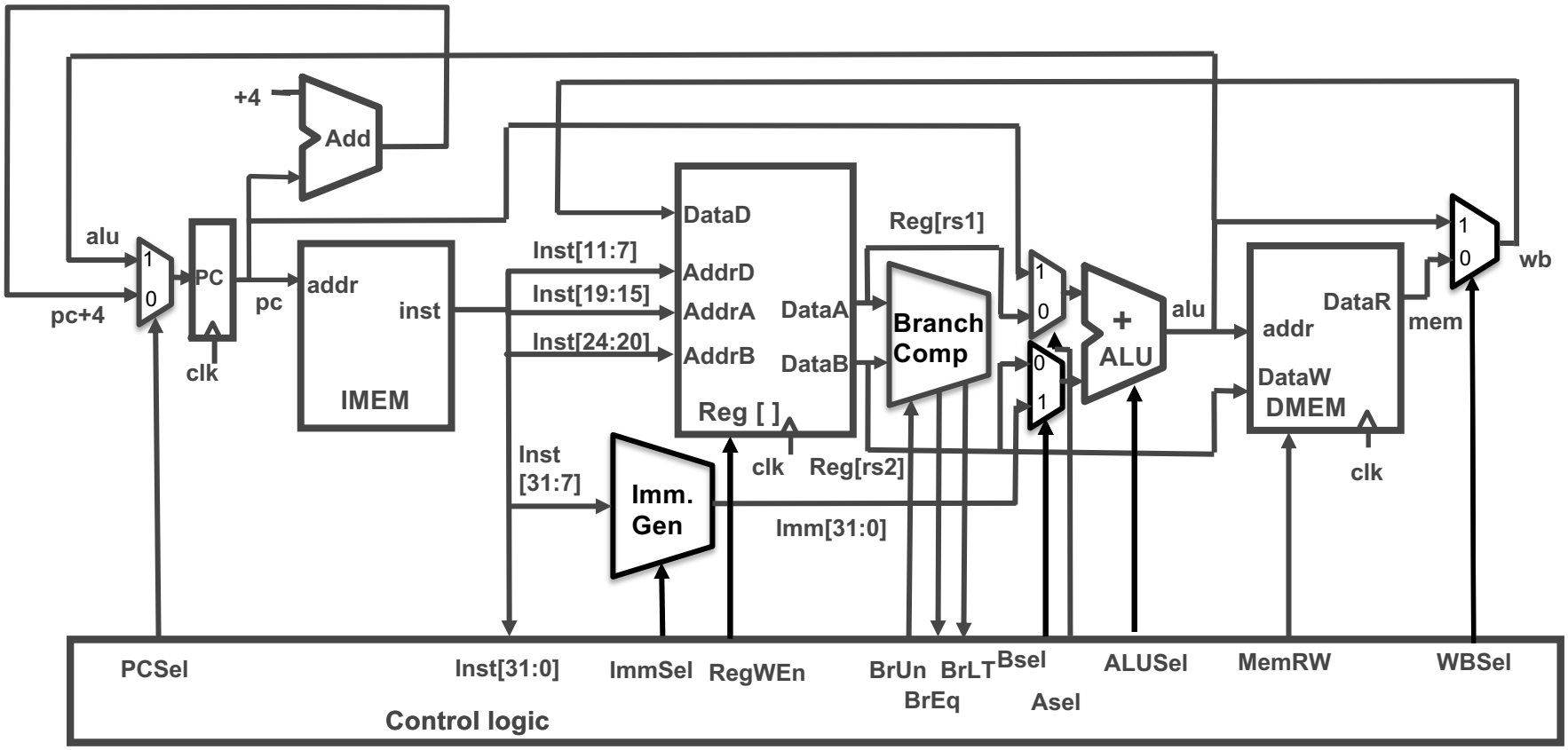


Let's Add JALR (I-Format)

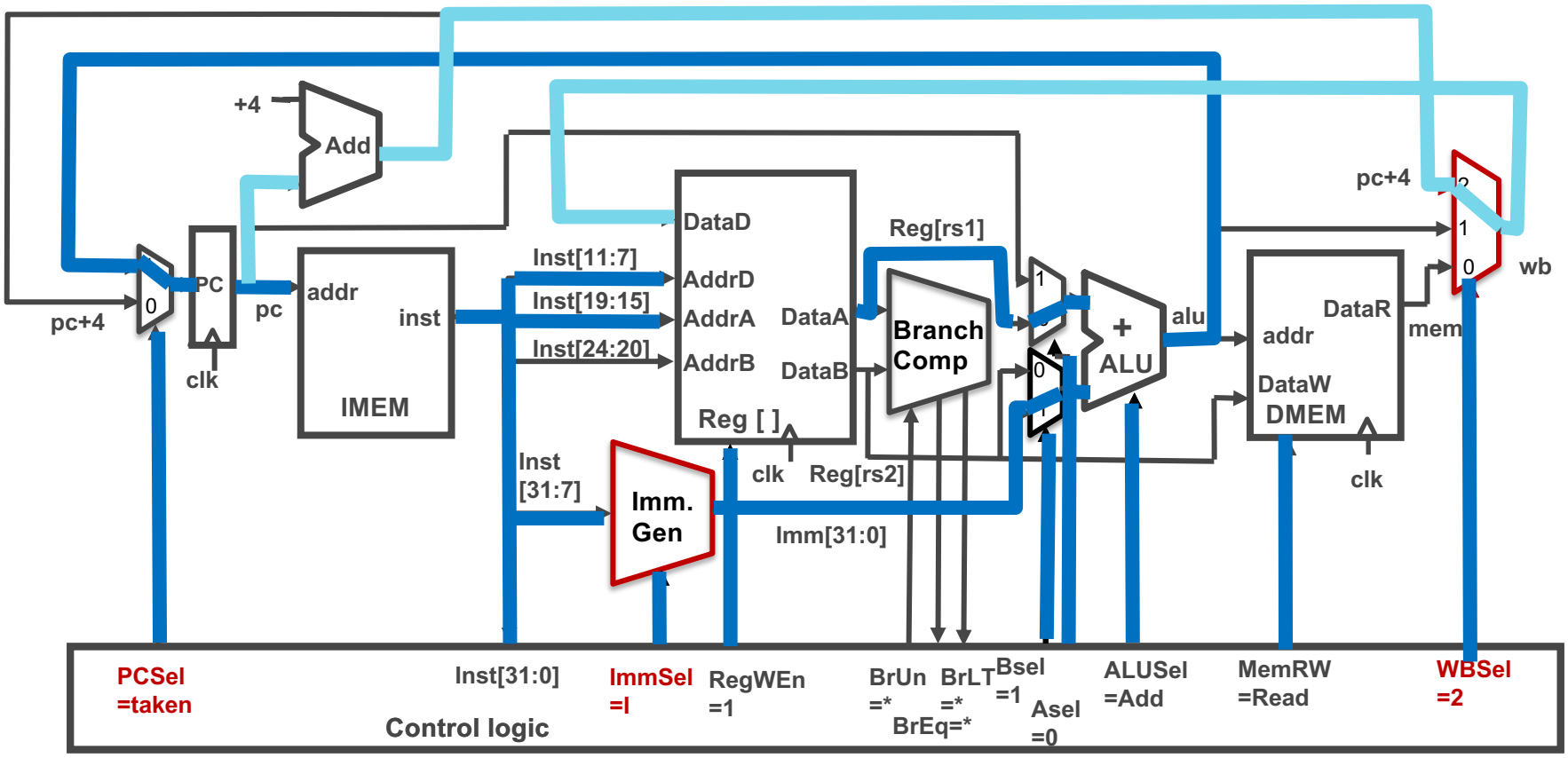


- JALR rd, rs, immediate
- Two changes to the state
  - Writes PC+4 to rd (return address)
  - Sets PC = rs + immediate
  - Uses same immediates as arithmetic and loads
    - ***no*** multiplication by 2 bytes
    - LSB is ignored

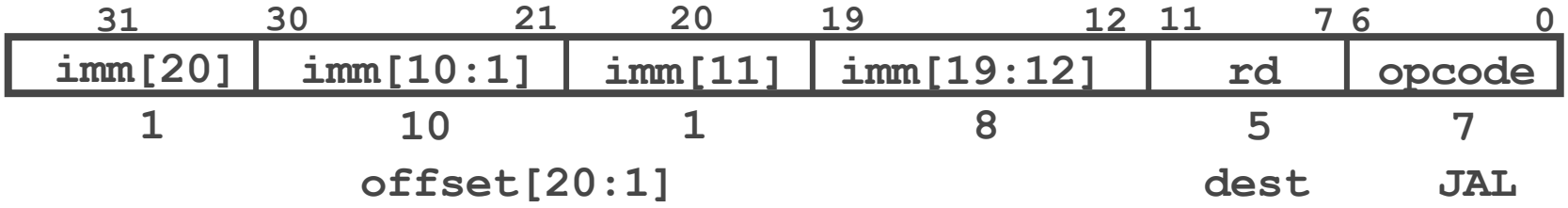
Datapath So Far, with Branches



Adding JALR

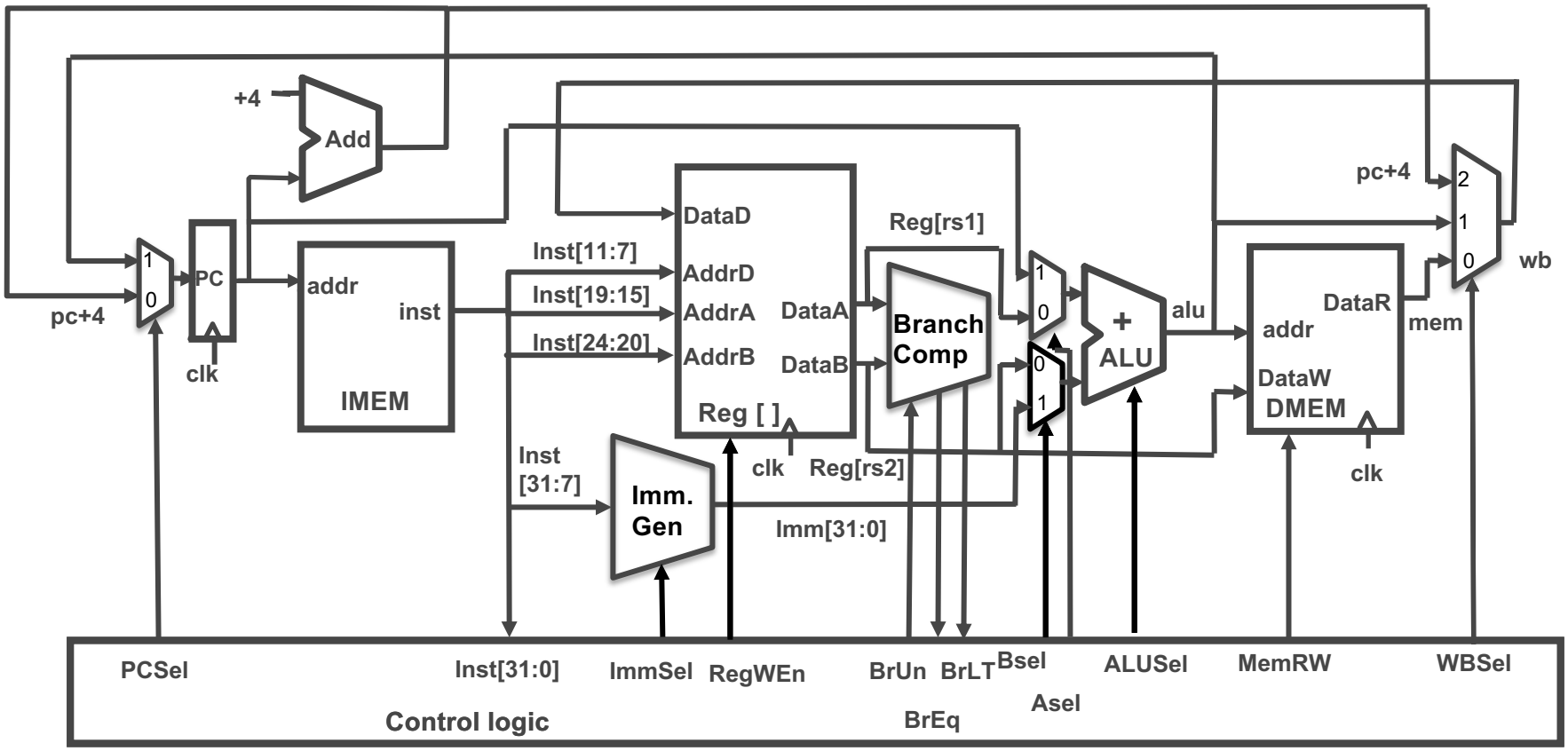

+4
Add
pc+4
0
PC
clk
pc
addr
inst
IMEM
Inst[11:7]
Inst[19:15]
Inst[24:20]
DataD
AddrD
AddrA
AddrB
DataA
DataB
Reg [ ]
clk
Reg[rs1]
Reg[rs2]
Branch Comp
1
0
+
ALU
alu
pc+4
1
0
wb
DataR
addr
DataW
DMEM
mem
clk
Inst [31:7]
Imm. Gen
Imm[31:0]
PCSel =taken
Inst[31:0]
Control logic
ImmSel =I
RegWEn =1
BrUn =*
BrEq=*
BrLT =*
Bsel =1
Asel =0
ALUSel =Add
MemRW =Read
WBSel =2

Adding JAL

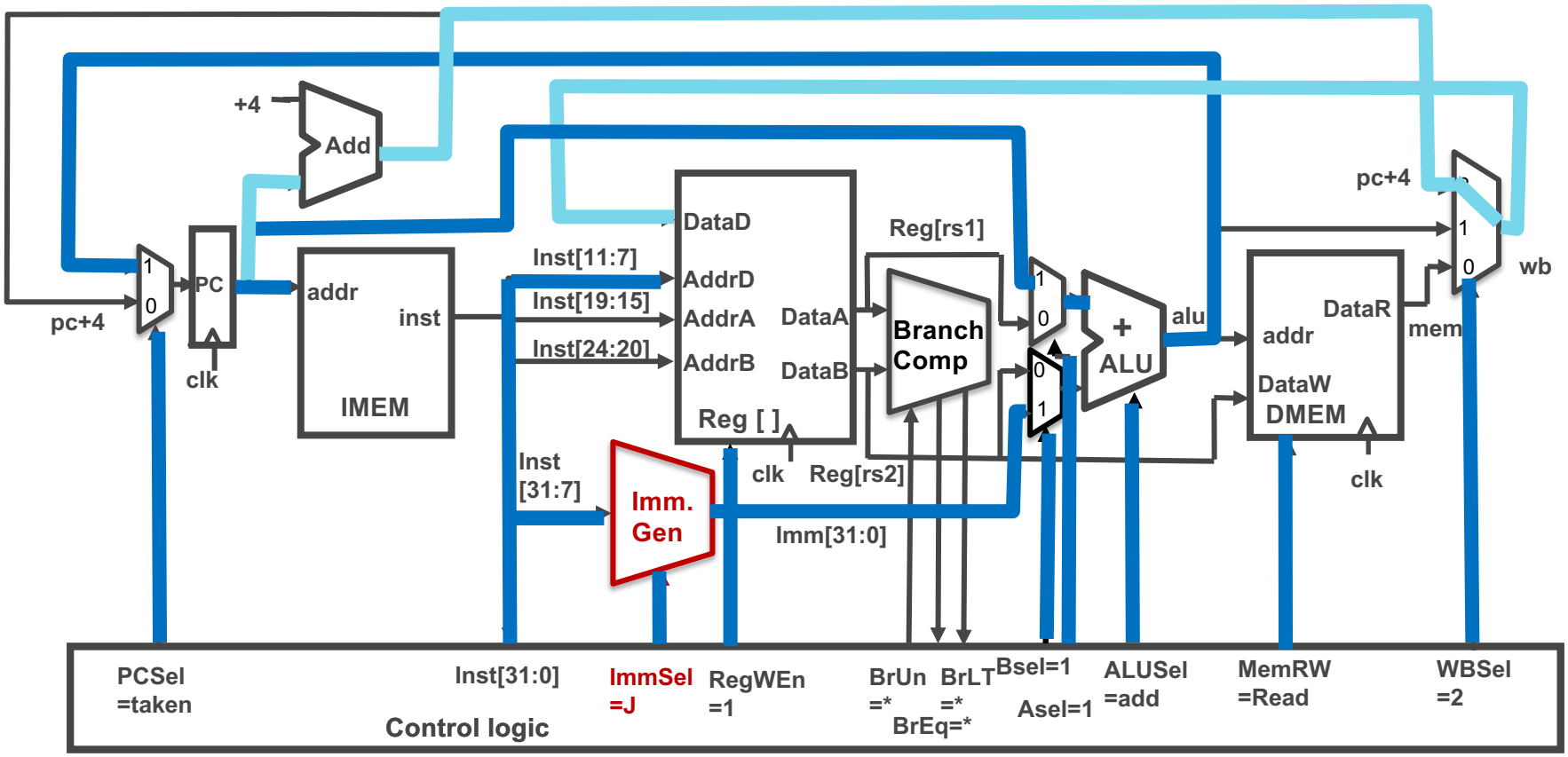


- JAL saves PC+4 in register rd (the return address)
- Set PC = PC + offset (PC-relative jump)
- Target somewhere within $\pm 2^{19}$ locations, 2 bytes apart
  - $\pm 2^{18}$ 32-bit instructions
- Immediate encoding optimized similarly to branch instruction to reduce hardware cost

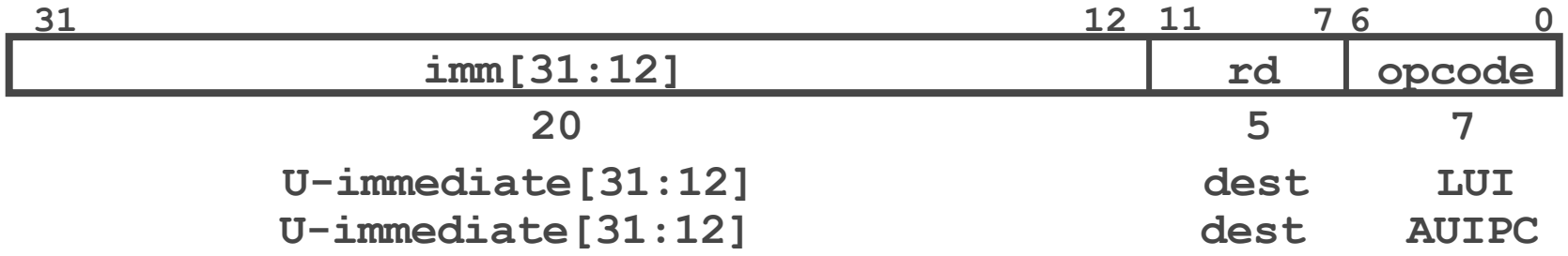
Datapath with JALR



Adding JAL

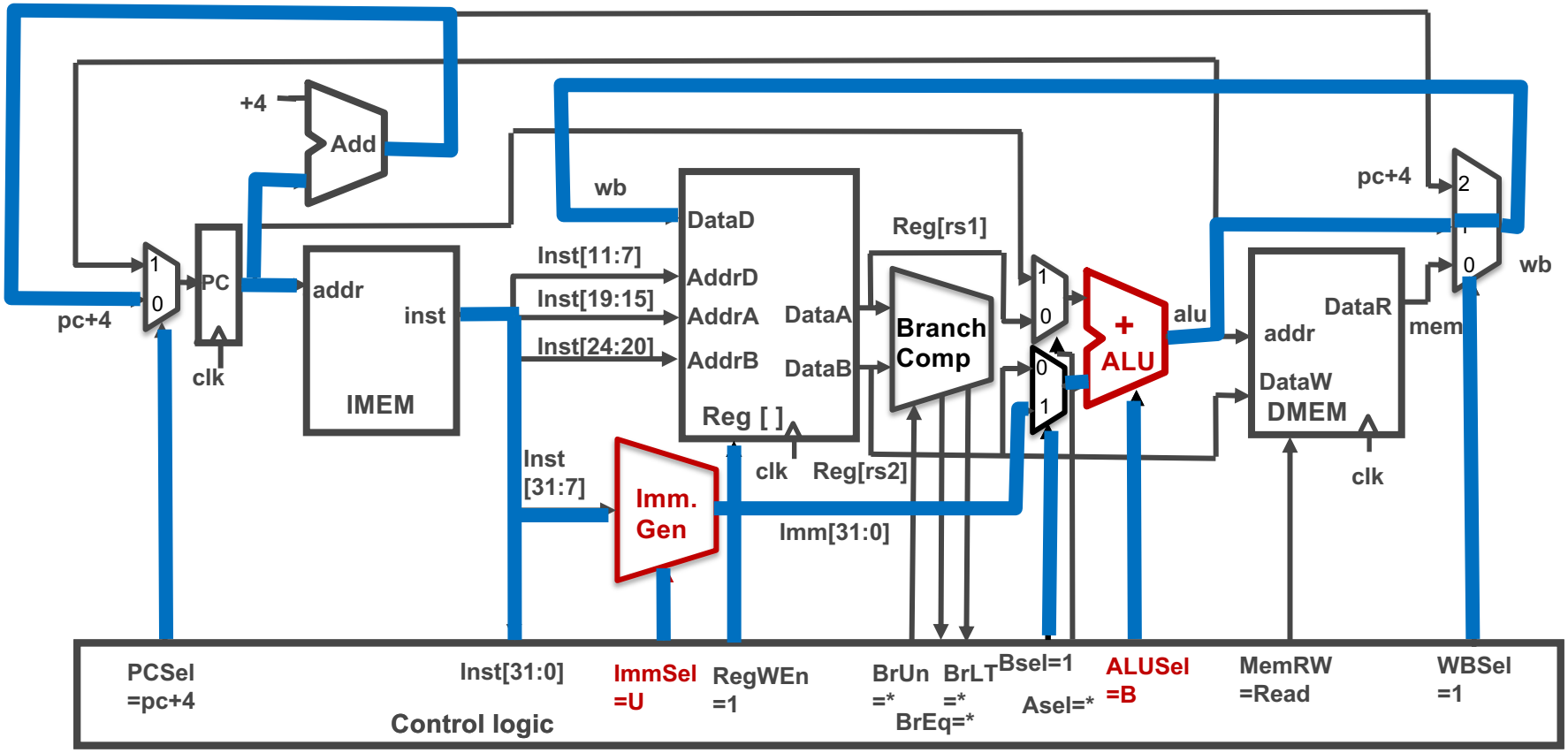

+4
Add
pc+4
1
0
PC
clk
addr
inst
IMEM
Inst[11:7]
Inst[19:15]
Inst[24:20]
DataD
AddrD
AddrA
AddrB
DataA
DataB
Reg [ ]
clk
Reg[rs1]
Reg[rs2]
Branch Comp
Inst [31:7]
Imm. Gen
Imm[31:0]
1
0
0
1
+
ALU
alu
pc+4
1
0
wb
DataR
addr
DataW
DMEM
clk
mem
PCSel =taken
Inst[31:0]
ImmSel =J
RegWEn =1
BrUn =*
BrLT =*
BrEq=*
Bsel=1
Asel=1
ALUSel =add
MemRW =Read
WBSel =2
Control logic

U-Format for “Upper Immediate” Instructions

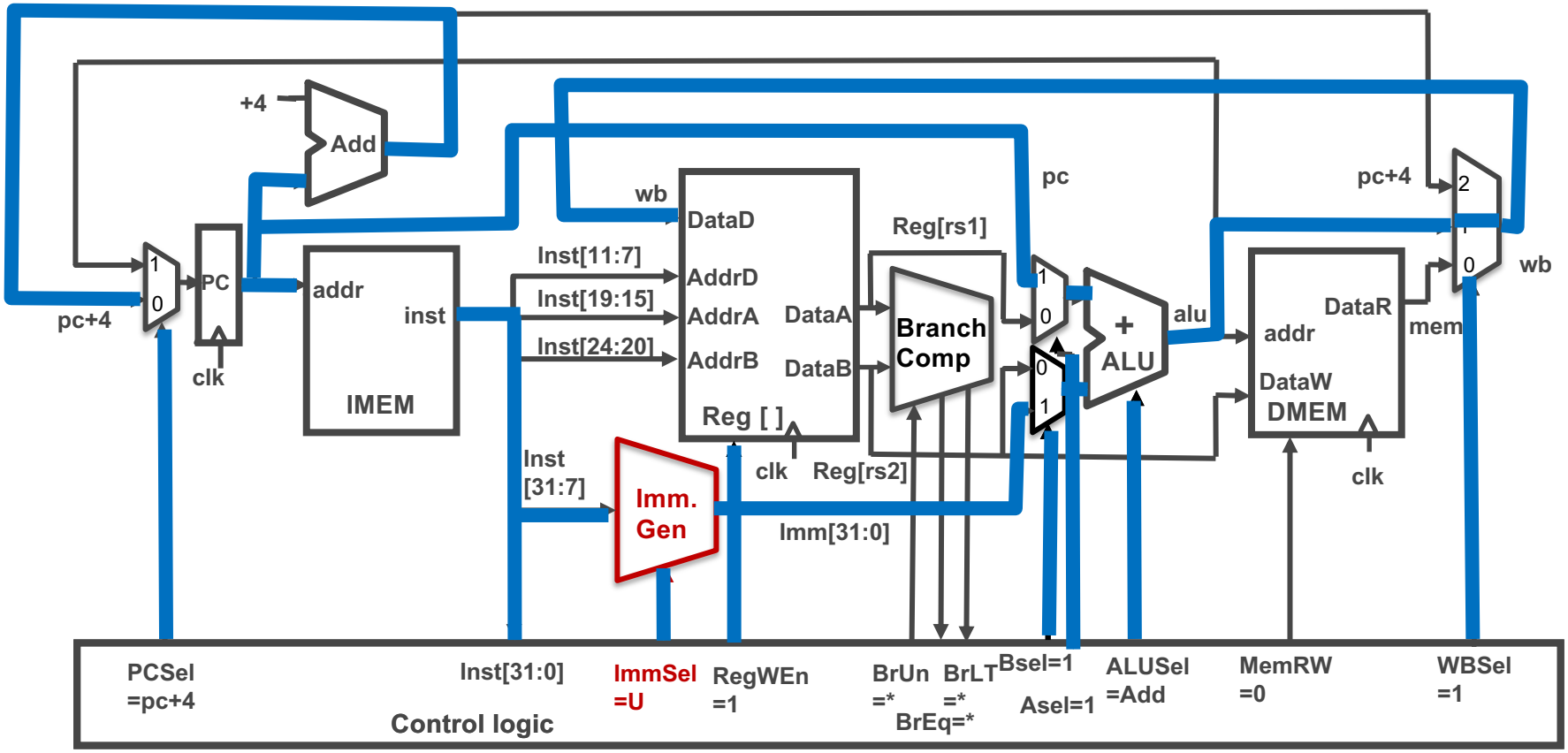


- Has 20-bit immediate in upper 20 bits of 32-bit instruction word
- One destination register, rd
- Used for two instructions
  - LUI – Load Upper Immediate
  - AUIPC – Add Upper Immediate to PC

Implementing LUI



Implementing AUIPC



Recap: Complete RV32I ISA

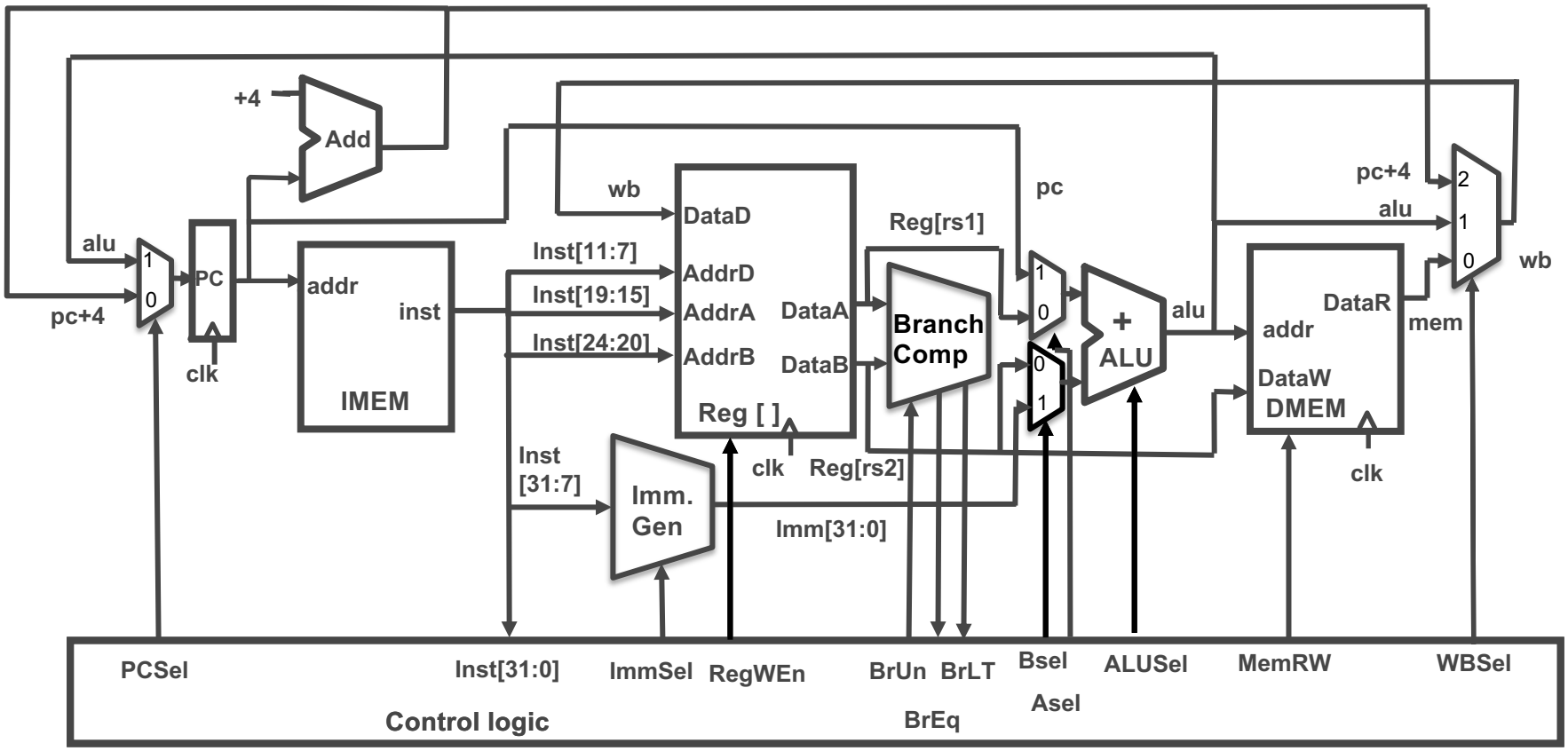
imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI

0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
0000 pred	succ	00000	000	00000	0001111	FENCE
0000 0000	0000	00000	001	00000	0001111	FENCE
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREA
csr		rs1	001	rd	1110011	CSRRW
csr		rs1	010	rd	1110011	CSRRS
csr		rs1	011	rd	1110011	CSRRC
csr		zimm	101	rd	1110011	CSRRW
csr		zimm	110	rd	1110011	CSRRSI
csr		zimm	111	rd	1110011	CSRRC

Not in CA

- RV32I has 47 instructions
- 37 instructions are enough to run any C program

Complete Single-Cycle RV32I Datapath!

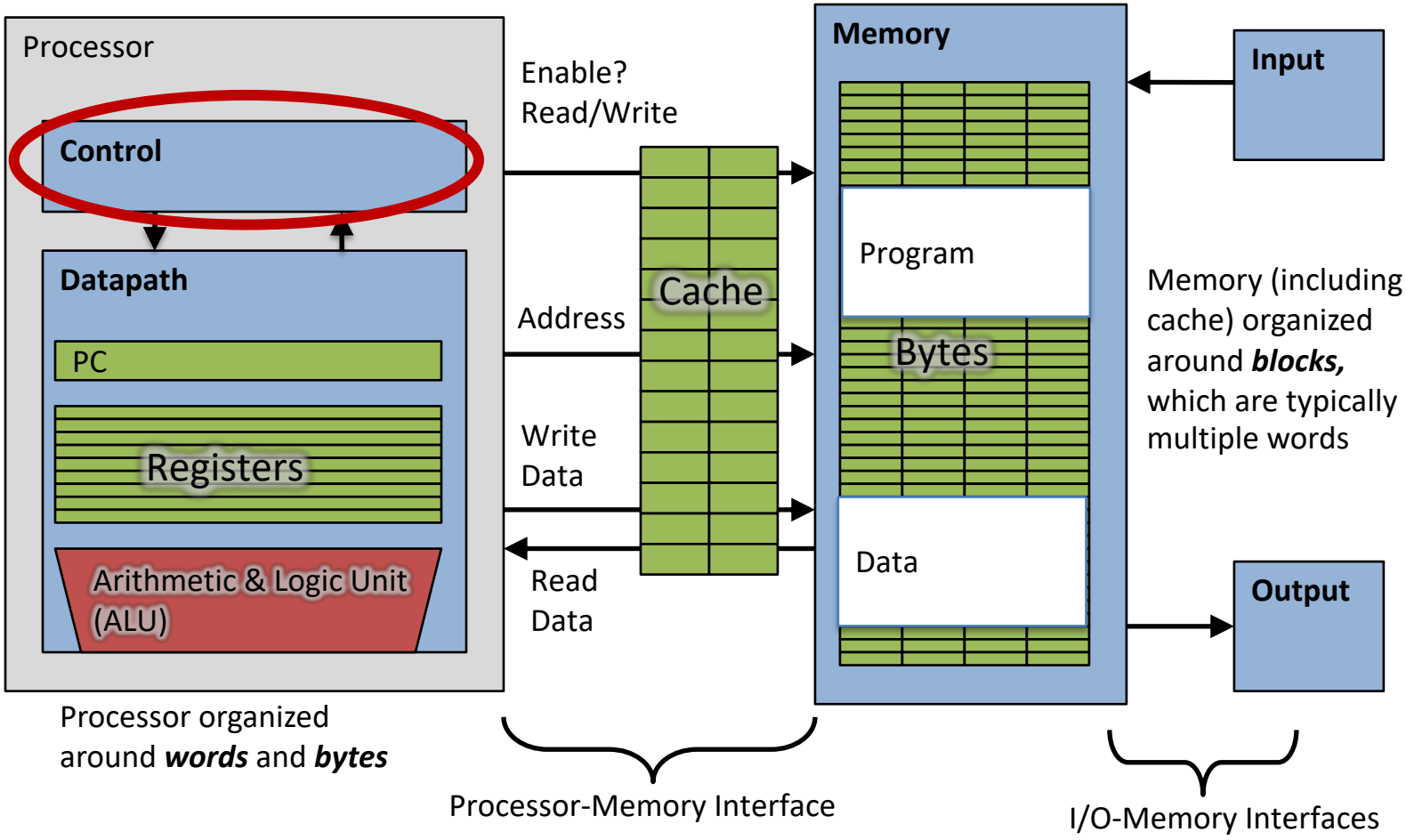


Register Transfer Level (RTL)

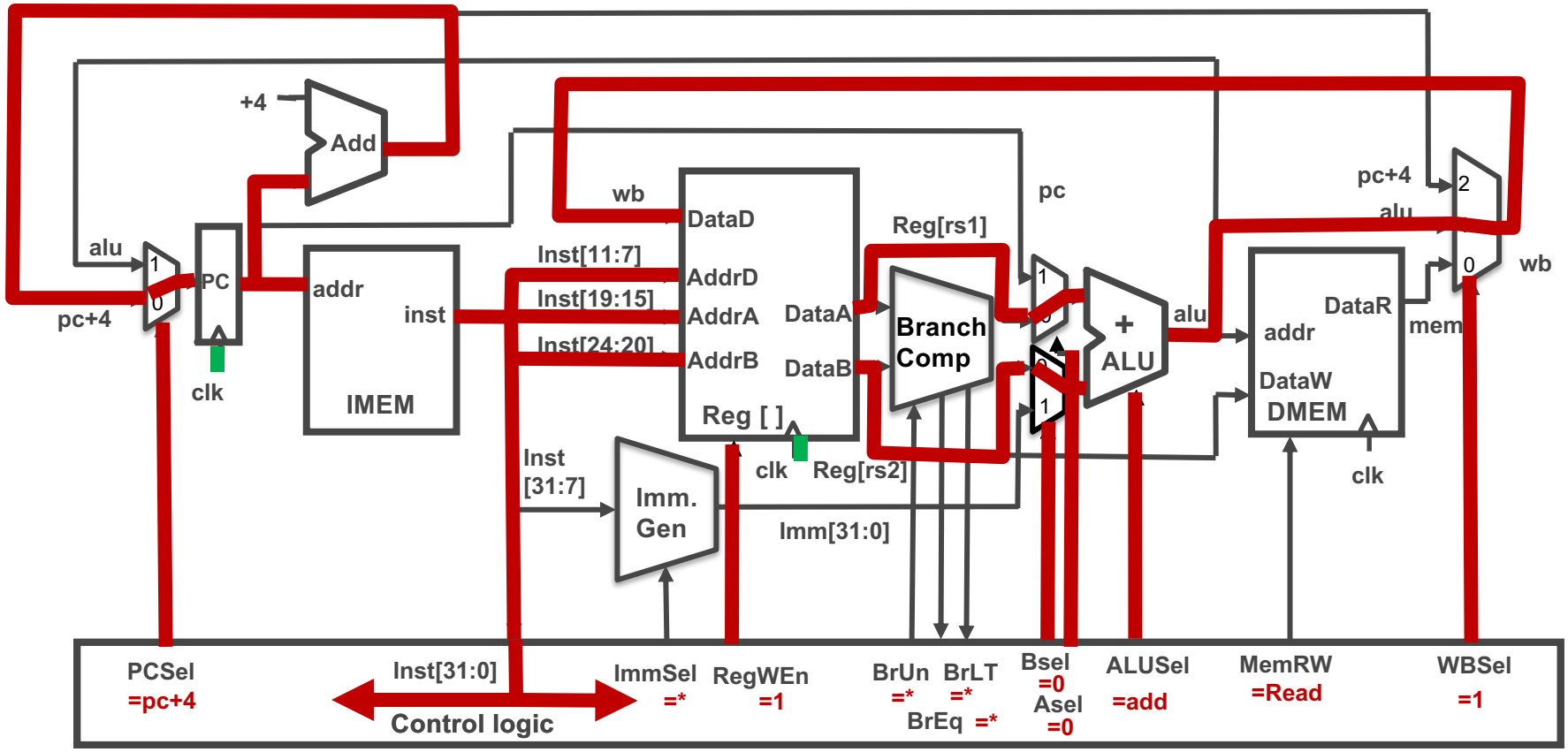
- RTL describes instructions in figures or text
- Can use C (or Verilog) to describe RTL

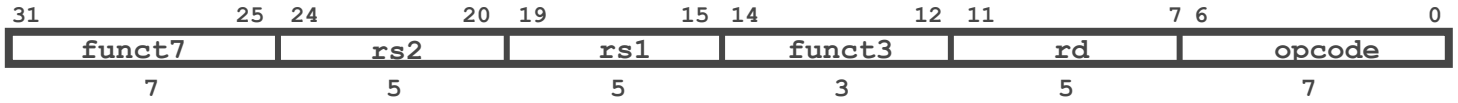
Inst	Register Transfers
add	R[rd] ← R[rs1] + R[rs2]; PC ← PC + 4
sub	R[rd] ← R[rs1] – R[rs2]; PC ← PC + 4
ori	R[rd] ← R[rs1] | Imm; PC ← PC + 4
jal	R[rd] ← PC + 4; PC ← R[rs1] + Imm
beq	if (R[rs1] == R[rs2]) PC ← PC + Imm else PC ← PC + 4

Our Single-Core Computer (without FPU, SIMD)

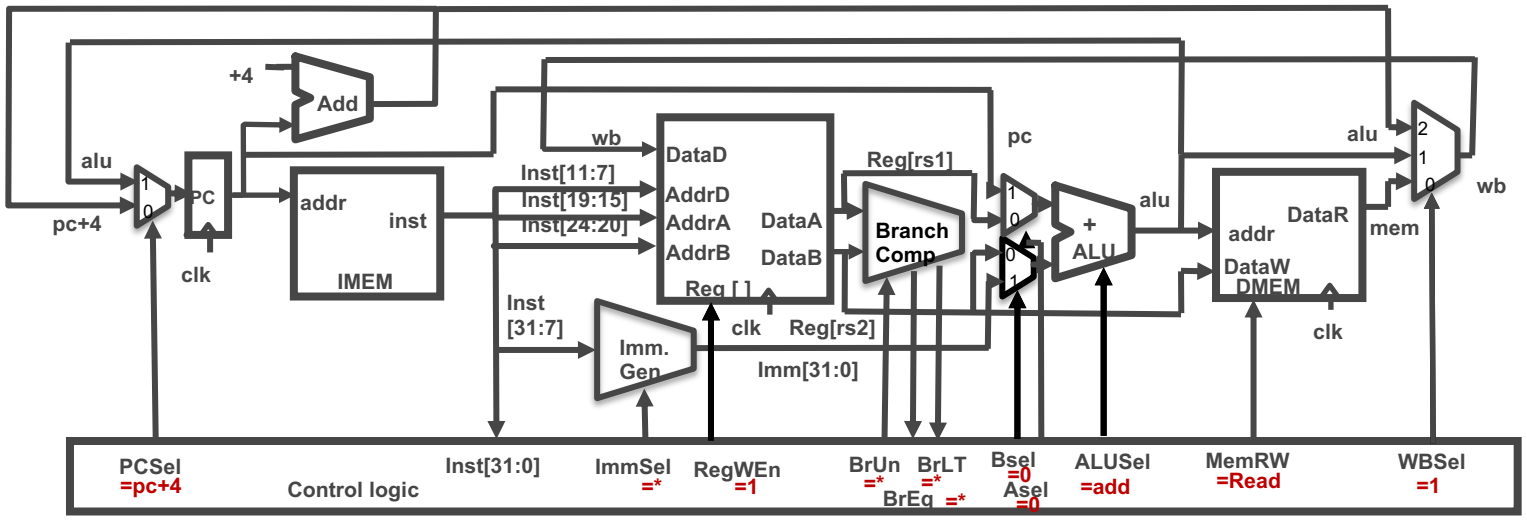


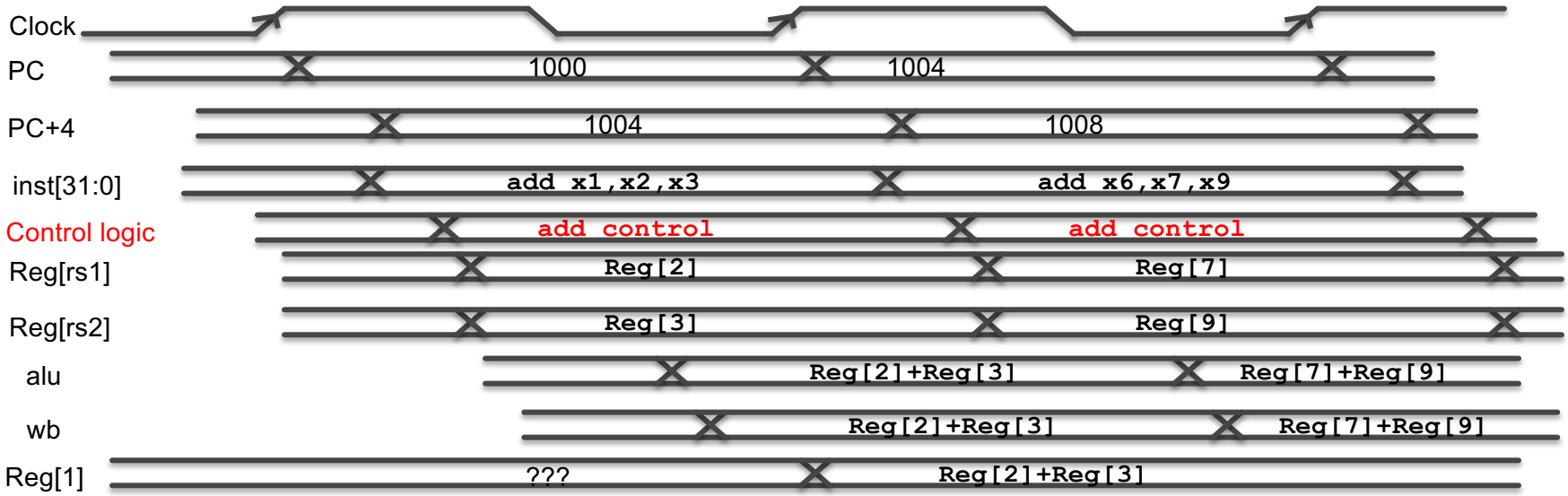
Example: add


+4
Add
alu
pc+4
PC
clk
addr
inst
IMEM
wb
DataD
Inst[11:7]
AddrD
Inst[19:15]
AddrA
DataA
Inst[24:20]
AddrB
DataB
Reg [ ]
clk
Reg[rs2]
Reg[rs1]
pc
Branch Comp
+
ALU
alu
pc+4
alu
2
0
wb
DataR
addr
mem
DataW
DMEM
clk
Inst [31:7]
Imm. Gen
Imm[31:0]
PCSel =pc+4
Inst[31:0]
Control logic
ImmSel =*
RegWEn =1
BrUn =*
BrLT =*
BrEq =*
Bsel =0
Asel =0
ALUSel =add
MemRW =Read
WBSel =1

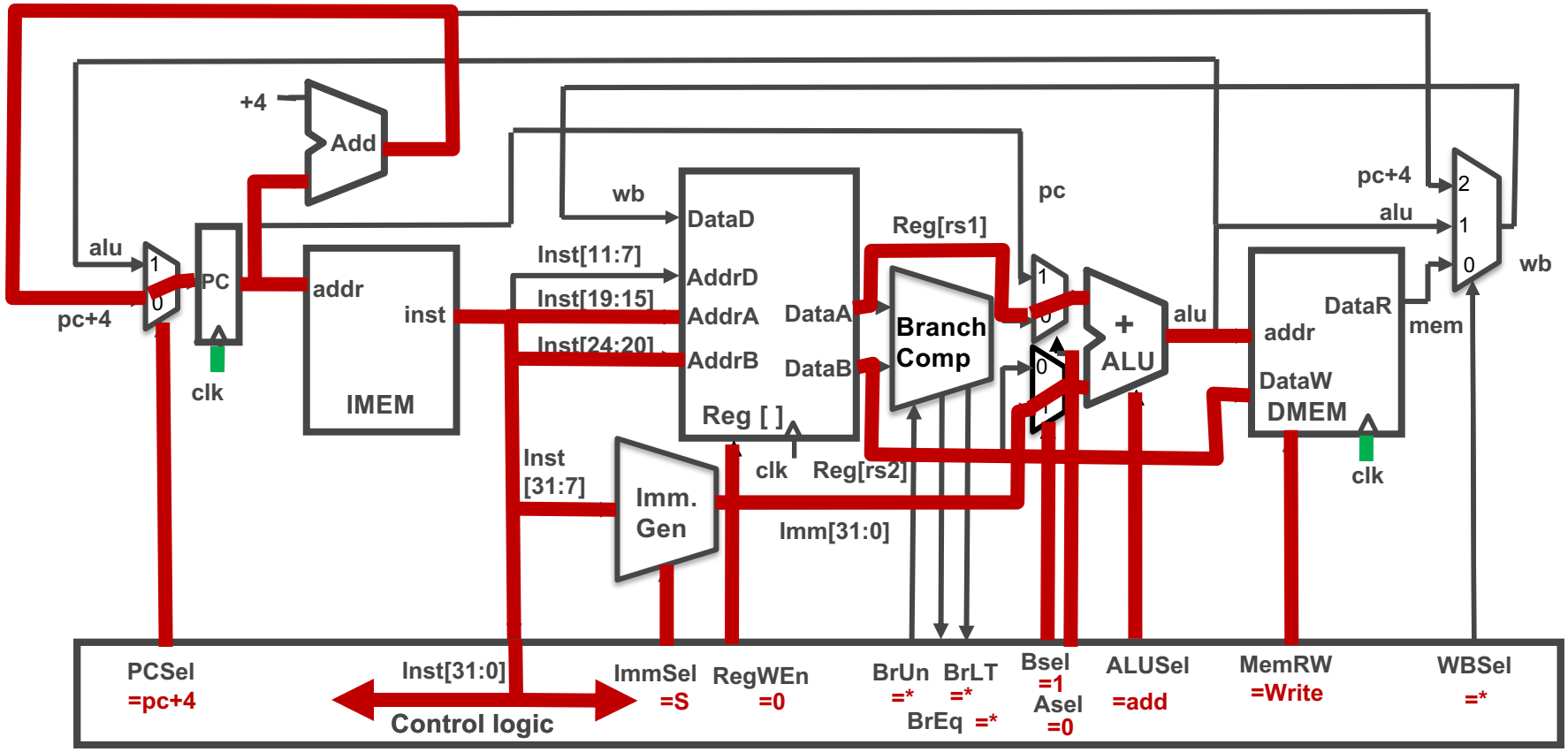

31
25
24
20
19
15
14
12
11
7
6
0
funct7
rs2
rs1
funct3
rd
opcode
7
5
5
3
5
7

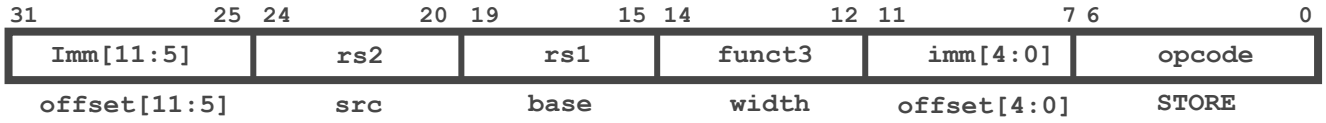
add Execution



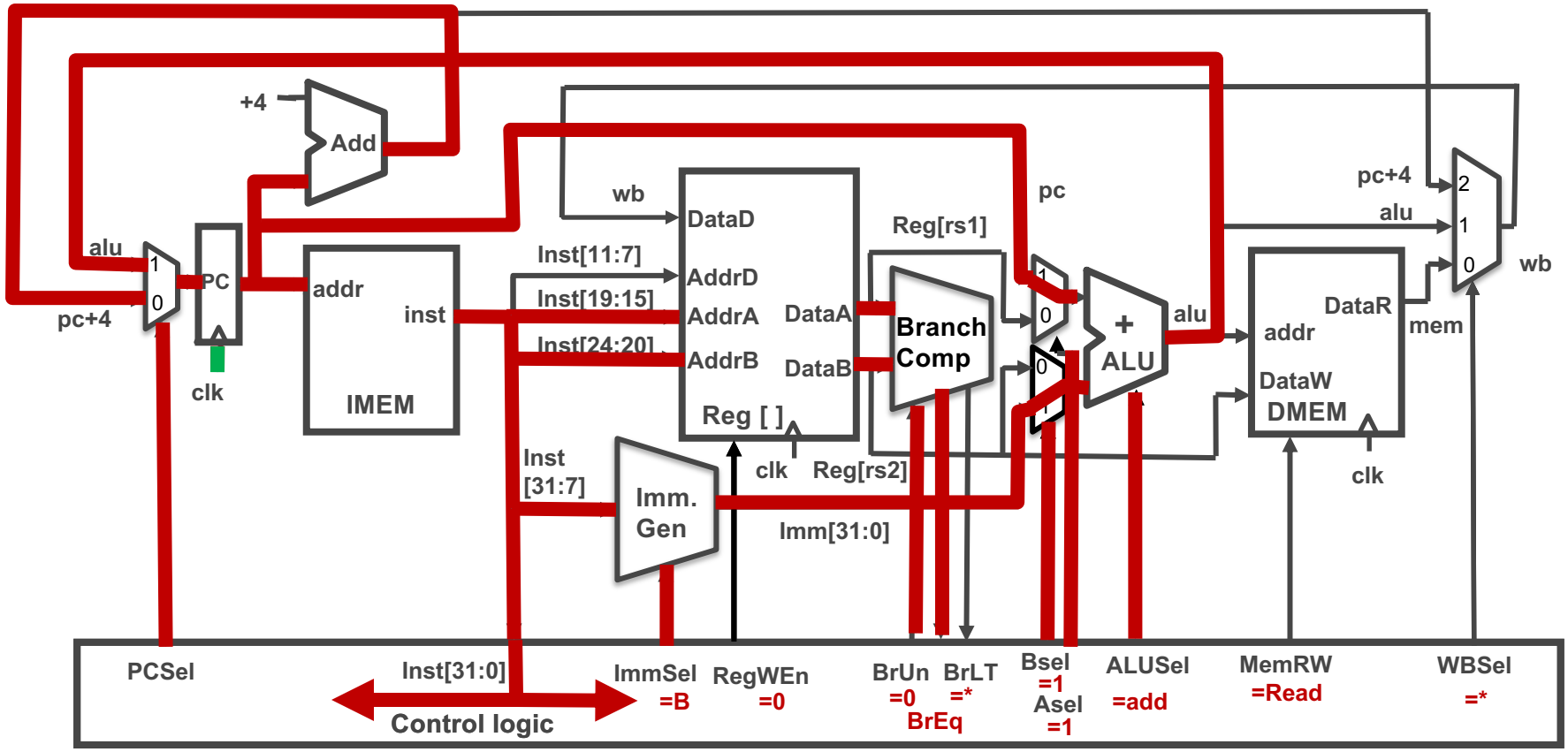


Example: sw


+4
Add
alu
pc+4
PC
clk
addr
inst
IMEM
wb
DataD
AddrD
AddrA
AddrB
DataA
DataB
Reg [ ]
clk
Reg[rs1]
Reg[rs2]
pc
Inst[11:7]
Inst[19:15]
Inst[24:20]
Inst
[31:7]
Imm.
Gen
Imm[31:0]
Branch
Comp
+
ALU
alu
addr
DataR
DataW
DMEM
mem
clk
pc+4
alu
wb
PCSel
=pc+4
Inst[31:0]
Control logic
ImmSel
=S
RegWEn
=0
BrUn
=*
BrLT
=*
BrEq =*
Bsel
=1
Asel
=0
ALUSel
=add
MemRW
=Write
WBSel
=*


31
25
24
20
19
15
14
12
11
7
6
0
Imm[11:5]
rs2
rs1
funct3
imm[4:0]
opcode
offset[11:5]
src
base
width
offset[4:0]
STORE

Example: beq


+4
Add
alu
1
0
pc+4
PC
clk
addr
inst
IMEM
wb
DataD
AddrD
AddrA
AddrB
DataA
DataB
Reg [ ]
clk
Reg[rs1]
Reg[rs2]
Inst[11:7]
Inst[19:15]
Inst[24:20]
Inst
[31:7]
Imm.
Gen
Imm[31:0]
pc
1
0
Branch
Comp
+
ALU
alu
pc+4
alu
2
1
0
wb
DataR
addr
DataW
DMEM
mem
clk
PCSel
Inst[31:0]
Control logic
ImmSel
=B
RegWEn
=0
BrUn
=0
BrEq
BrLT
=*
Bsel
=1
Asel
=1
ALUSel
=add
MemRW
=Read
WBSel
=*

Midterm I

- Date: Wednesday, Apr. 03
- Time: 08:15- 9:55 (normal lecture slot)
  - Be punctual – we start 8:15 sharp!
- Venue: Teaching Center 301 + 201
- Closed book:
  - You can bring **one** A4 page with notes (both sides; in English): Write your Chinese and **Pinyin** name on the top! **Handwritten** by you!
  - You will be provided with the RISC-V ”green sheet”
  - No other material allowed!

Midterm I

- Switch cell phones **off**! (not silent mode – off!)
  - Put them in your bags.
- Bags under the table. Nothing except paper, pen, 1 drink, 1 snack on the table!
- No other electronic devices are allowed!
  - No ear plugs, music, smartwatch…
- Anybody touching any electronic device will **FAIL** the course!
- Anybody found cheating (copy your neighbors answers, additional material, ...) will **FAIL** the course!


CASIO
JF-120TM
TWO WAY POWER
12 DIGITS
F CUT 5/4
4 3 2 1 0 A
TAX–
TAX RATE
TAX+
RATE SET
%
GT
MRC
ON
AC
C
7
8
9
4
5
6
1
2
3
0
00
=
+
–
×


08:08
Thu, Sep 1
HUAWEI


Tue
9


COMPUTER
ORGANIZATION
AND DESIGN
THE HA WARE/SOFTWA INTERFACE
RISC-V EDITION
MK
MORGAN KAUFMANN
DAVID A. PATTERSON
JOHN L. HENNESSY

FUNCTIONS OF SEVERAL VARIABLES

$z = f(x,y)$ $w = f(x,y,z)$

LEVEL CURVES (one down) $f(x,y) = k$ = CONST. CONTOUR MAPS (2-D)

$w = f(x,y,z) = k$ = CONST. SURFACE LAYERS (3-D)

FUNCTION OF n VARIABLES

$z = f(x_1, x_2, \dots, x_n)$

$\mathbb{R}^n \longrightarrow \mathbb{R}$ 3 WAYS TO LOOK AT THIS FUNCTION

1. As a function of n real variables $x_1, x_2, \dots x_n$
2. As a function of a single pt-variable $(x_1, \dots x_n)$
3. As a function of a single vector var. $\vec{x} = \langle x_1, \dots x_n \rangle$

PARTIAL DERIVATIVES

Derivatives w/ Respect to one variable, while holding the other variables constant

$z = f(x,y)$ NOTATIONS

$f_x(x,y) = f_x = \frac{\partial f}{\partial x} = \frac{\partial f(x,y)}{\partial x} = \frac{\partial z}{\partial x}$

$f_y(x,y) = f_y = \frac{\partial f}{\partial y} = \frac{\partial f(x,y)}{\partial y} = \frac{\partial z}{\partial y}$

SAME HOLDS FOR FUNCTIONS OF MORE THAN TWO VARIABLES

SECOND PARTIAL DERIVATIVES

$f_{xx} = \frac{\partial}{\partial x}\left(\frac{\partial f}{\partial x}\right) = \frac{\partial^2 f}{\partial x^2} = \frac{\partial^2 z}{\partial x^2} = \frac{\partial}{\partial x}\left(\frac{\partial z}{\partial x}\right)$

$f_{xy} = \frac{\partial}{\partial y}\left(\frac{\partial f}{\partial x}\right) = \frac{\partial^2 f}{\partial y \partial x} = \frac{\partial^2 z}{\partial y \partial x} = \frac{\partial}{\partial y}\left(\frac{\partial z}{\partial x}\right)$

$f_{yx} = \frac{\partial}{\partial x}\left(\frac{\partial f}{\partial y}\right) = \frac{\partial^2 f}{\partial x \partial y} = \frac{\partial^2 z}{\partial x \partial y} = \frac{\partial}{\partial x}\left(\frac{\partial z}{\partial y}\right)$

$f_{yy} = \frac{\partial}{\partial y}\left(\frac{\partial f}{\partial y}\right) = \frac{\partial^2 f}{\partial y^2} = \frac{\partial^2 z}{\partial y^2} = \frac{\partial}{\partial y}\left(\frac{\partial z}{\partial y}\right)$

CLAIRAUT'S THEOREM

IF f_{xy} AND f_{yx} ARE BOTH CONTINUOUS

$f_{xy}(a,b) = f_{yx}(a,b)$

PARTIAL DIFF. EQ'S

LAPLACE'S EQUATION $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$ etc...

THE WAVE EQUATION $\frac{\partial^2 u}{\partial t^2} = a^2 \frac{\partial^2 u}{\partial x^2}$ etc...

THE CHAIN RULE

SINGLE VARIABLE $y = f(u)$, $u = g(x)$, ie $y = f(g(x))$

$y'(x) = f'(g(x)) \cdot g'(x)$ $\quad \frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$

CASE 1 $z = f(x,y)$, $x = g(t)$, $y = h(t)$ ie $z = f(g(t), h(t))$

$\frac{dz}{dt} = \frac{\partial z}{\partial x}\frac{dx}{dt} + \frac{\partial z}{\partial y}\frac{dy}{dt}$ or w/ $z = f$ $\frac{\partial z}{\partial t} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial t}$ SAME PARTIAL

CASE 2 $z = f(x,y)$, $x = g(s,t)$, $y = h(s,t)$; ie $z = f(g(s,t), h(s,t))$

$\frac{\partial z}{\partial s} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial s} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial s}$ $\quad \frac{\partial z}{\partial t} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial t}$

THINGS APPEAR TO CANCEL EVEN THOUGH $\partial \neq d$ BUT DON'T DO THAT

CHAIN RULE: GENERAL VERSION

u is a function of $t_1, \dots t_m$ $\quad u = f(x_1, \dots x_n)$ $\quad x_i = g(t_1, \dots t_m)$ $\quad i = 1,2,\dots n$

$\frac{\partial u}{\partial t_i} = \frac{\partial u}{\partial x_1}\frac{\partial x_1}{\partial t_i} + \frac{\partial u}{\partial x_2}\frac{\partial x_2}{\partial t_i} + \dots + \frac{\partial u}{\partial x_n}\frac{\partial x_n}{\partial t_i}$ for each $i = 1,2,\dots m$

IMPLICIT DIFFERENTIATION

You can always solve for y' and diff. implicitly

$\frac{dy}{dx} = -\frac{\frac{\partial F}{\partial x}}{\frac{\partial F}{\partial y}} = -\frac{F_x}{F_y}$ $\quad \frac{\partial z}{\partial x} = -\frac{\frac{\partial F}{\partial x}}{\frac{\partial F}{\partial z}} = -\frac{F_x}{F_z}$ $\quad \frac{\partial z}{\partial y} = -\frac{\frac{\partial F}{\partial y}}{\frac{\partial F}{\partial z}} = -\frac{F_y}{F_z}$

$F(x,y) = 0$ $y = f(x)$, $F(x,f(x)) = 0$ $\quad F(x,y,z) = 0$ $z = f(x,y)$, $F(x,y,f(x,y)) = 0$

TANGENT PLANE TO A LEVEL SURFACE

$\nabla F \perp$ TO TAN. VECTOR

$F_x(x - x_0) + F_y(y - y_0) + F_z(z - z_0) = 0$ ACCELERATION at (x_0, y_0, z_0) $\quad \nabla F \cdot \vec{r}'(t) = 0$

NORMAL LINE TO A LEVEL SURFACE

$\frac{x - x_0}{F_x} = \frac{y - y_0}{F_y} = \frac{z - z_0}{F_z}$ ALL @ (x_0, y_0, z_0)

SPECIAL CASE $z = f(x,y)$ $\quad F(x,y,z) = f(x,y) - z = 0$ LEVEL SURFACE W/ K=0 OLD DEFINITION

THEN $F_z = -1$, $\nabla F = \langle f_x, f_y, -1 \rangle$ and TAN PLANE IS GIVEN BY $z - z_0 = f_x(x - x_0) + f_y(y - y_0)$

MAXIMUM AND MINIMUM VALUES $z = f(x,y)$

$f_x(a,b) = 0$ & $f_y(a,b) = 0$ $\Rightarrow \nabla f(a,b) = \langle 0,0 \rangle = \vec{0}$ NECESSARY BUT NOT SUFFICIENT TO GUARANTEE A MAX. OR MIN.

① Set $f_x = f_y = 0$ Solve For Critical Pts. (Always check $(0,0)$ the origin) THEN APPLY THE 2ND DERIVATIVE TEST Find $f_{xx}, f_{yy}, f_{xy} = f_{yx}$

$D = \begin{vmatrix} f_{xx} & f_{xy} \\ f_{yx} & f_{yy} \end{vmatrix} = f_{xx}f_{yy} - (f_{xy})^2$ • $D > 0$, $f_{xx} > 0$ LOCAL MIN. • $D > 0$, $f_{xx} < 0$ LOCAL MAX • $D < 0$ SADDLEPT. • $D = 0$ NO INFO

FINDING ABSOLUTE MAX. AND MINS. FOR f ON A CLOSED, BOUNDED SET

1. Find values of f at the critical points of f in D
2. Find the extreme values of f on the Boundary of D
3. The largest value from 1, 2 is the ABS. MAX, the smallest is the ABS. MIN

MAXIMIZING AND MINIMIZING Set up a function of two variables of the form $z = f(x,y)$ and then Do the Usual Routine

DOMAINS: Allowed (x,y), (x,y,z) RANGES: $\mathbb{R}$ SW's

ε-δ DEFINITION OF CONTINUITY

LET f BE A FUNCTION OF 2 VARIABLES DEFINED ON A DISK W/ CENTER (a,b), EXCEPT POSSIBLY @ (a,b), THEN $\lim_{(x,y)\to(a,b)} f(x,y) = L$

IF FOR EVERY $\# \varepsilon > 0$, THERE IS A CORRESPONDING $\# \delta > 0$ ST. $|f(x,y) - L| < \varepsilon$ whenever $0 < \sqrt{(x-a)^2 + (y-b)^2} < \delta$

IF THE LIMIT AS A FUNCTION APPROACHES A POINT (a,b) ALONG TWO DIFFERENT PATHS IS NOT THE SAME, THE LIMIT DOES NOT EXIST ⊗

① $f(x,y)$ IS CONTINUOUS AT (a,b) IF THE LIMIT OF $f(x,y)$ AS $(x,y) \to (a,b)$ EXISTS.

② COMPOSITE FUNCTIONS OF CONTINUOUS FUNCTIONS ARE CONTINUOUS, AS ARE SUMS AND PRODUCTS

EQUATIONS OF TANGENT PLANES TO SURFACES

$z = f(x,y)$ @ (x_0, y_0, z_0) EVAUATED AT A POINT

$z - z_0 = f_x(x_0, y_0)(x - x_0) + f_y(x_0, y_0)(y - y_0)$

TOTAL DIFFERENTIAL $(dy = f'(x)dx$ SINGLE VARIABLE)

$dz = f_x(x,y)\,dx + f_y(x,y)\,dy = \frac{\partial z}{\partial x}dx + \frac{\partial z}{\partial y}dy$

INCREMENTS $\Delta x, \Delta y, \Delta z$ DIFFERENTIALS dx, dy, dz

FOR SMALL $\Delta x, \Delta y$ $\quad \Delta x = dx$, $\Delta y = dy$

② IF f_x AND f_y ARE CONTINUOUS $\Delta z \approx dz$

(ie change in height of surface (Δz) $\approx$ change in height of the tangent plane (dz))

$\Delta z = f(a + \Delta x, b + \Delta y) - f(a,b)$ THEOREM

$\Delta z = f_x(a,b)\Delta x + f_y(a,b)\Delta y + \varepsilon_1 \Delta x + \varepsilon_2 \Delta y$ where ε_1 and ε_2 are functions of Δx and Δy that Approach 0 as $(\Delta x, \Delta y) \to (0,0)$

$\Rightarrow f$ is DIFFERENTIABLE @ (a,b) for $z = f(x,y)$ DEF.

DEPENDENCY DIAGRAMS CASE 1, 2

You can find derivatives for all the fundamental variables

$\partial x, \partial y$ Have NO MEANING LIKE dx and dy Do. INDEPENDENT

THE GRADIENT VECTOR $z = f(x,y)$ AT A POINT

$\nabla f(x,y) = \left(\frac{\partial f}{\partial x}(x,y), \frac{\partial f}{\partial y}(x,y)\right) = \langle f_x, f_y \rangle = \left\langle \frac{\partial z}{\partial x}, \frac{\partial z}{\partial y} \right\rangle$

DIRECTIONAL DERIVATIVES $D_{\vec{u}} f$, $\vec{u} = \langle a,b \rangle$

$D_{\vec{u}} f(x,y) = f_x(x,y)\,a + f_y(x,y)\,b$ SAME FOR 3 VARIABLES

$D_{\vec{u}} f(x,y) = \nabla f(x,y) \cdot \vec{u}$

$D_{\vec{u}} f$ max occurs when ∇f is in the same Dir. as $\vec{u}$

$D_{\vec{u}} f = \nabla f \cdot \vec{u} = |\nabla f||\vec{u}| \cos\theta = |\nabla f| \quad (|\vec{u}| = 1, \theta = 0)$

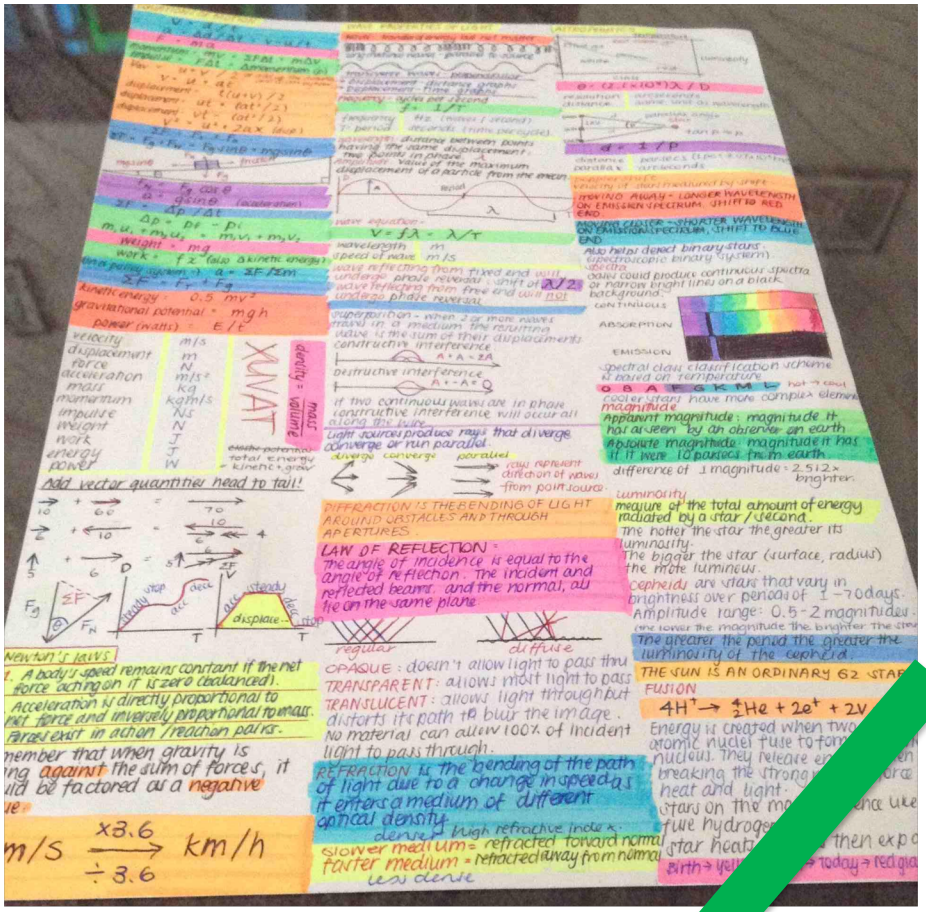
③ THE GRADIENT VECTOR POINTS IN THE DIRECTION OF STEEPEST ASCENT OR DESCENT (on a surface)

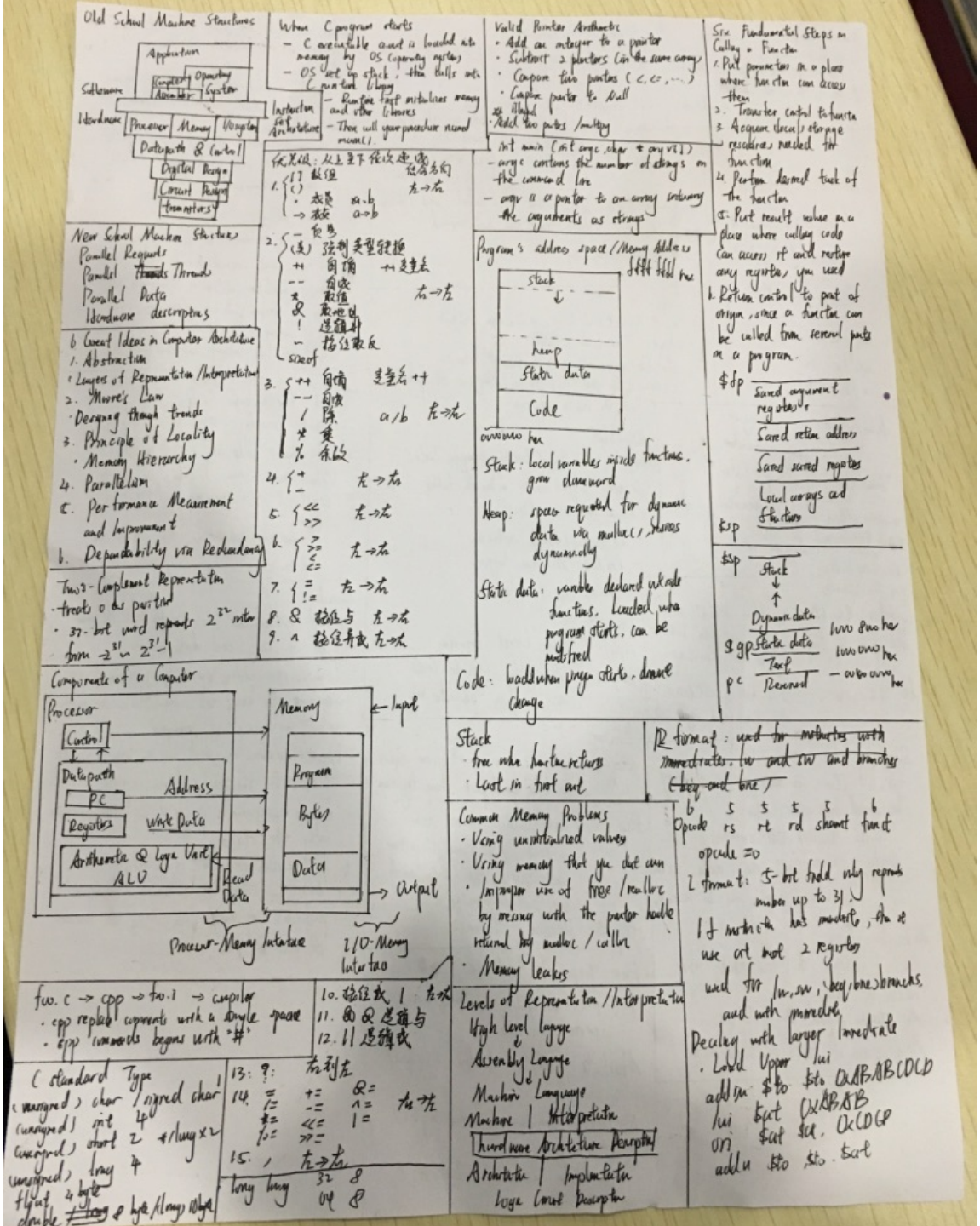
④ THE GRADIENT VECTOR IS ORTHOGONAL TO THE LEVEL CURVES OF A SURFACE

⑤ ∇f HAS AS MANY COMPONENTS AS f HAS INDEPENDENT VARIABLES. $\nabla^n \langle f_{x_1}, f_{x_2}, \dots f_{x_n} \rangle$ (n-vector)

⑥ TO FIND THE NORMAL (AND LATER TANGENT PLANE) TO A SURFACE, LET THAT SURFACE BE THE LEVEL SET OF SOME HIGHER DIMENSIONAL FUNCTION. THEN THE GRADIENT OF THE HIGHER D FUNCTION IS $\perp$ TO YOUR SURFACE

Ex ? $\vec{n}$ to $x^2 + y^2 + z^2 = 1$ Let $W = x^2 + y^2 + z^2 - 1$ $x^2 + y^2 + z^2 = 1$ IS THE LEVEL SET $W = 0$ SO $\nabla W = \langle 2x, 2y, 2z \rangle$ IS A NORMAL VECTOR TO THE 3-D SPHERE $x^2 + y^2 + z^2 = 1$




Old School Machine Structures
Application
Operating System
Compiler
Assembler
Software
Hardware
Processor
Memory
Input/Output
Datapath & Control
Digital Design
Circuit Design
Transistors
When C program starts
- C executable exact is loaded into memory by OS (operating system)
- OS set up stack, then calls into C runtime library
- Runtime init initializes memory and other libraries
- Then will your procedure named main()
Instruction Set Architecture
Valid Pointer Arithmetic
• Add an integer to a pointer
• Subtract 2 pointers (in the same array)
• Compare two pointers (<, <=, ...)
• Compare pointer to NULL
• Add two pointers (illegal)
Six Fundamental Steps in Calling a Function
1. Put parameters in a place where function can access them
2. Transfer control to function
3. Acquire (local) storage resources needed for function
4. Perform desired task of the function
5. Put result value in a place where calling code can access it and restore any registers you used
6. Return control to point of origin, since a function can be called from several points in a program.
int main (int argc, char * argv[])
- argc contains the number of strings on the command line
- argv is a pointer to an array containing the arguments as strings
New School Machine Structures
Parallel Requests
Parallel Threads
Parallel Data
Hardware descriptions
Program's address space (Memory Address)
stack
heap
Static data
Code
0000 0000 hex
FFFF FFFF hex
Stack: local variables inside functions, grow downward
Heap: space requested for dynamic data via malloc(); Stores dynamically
Static data: variable declared outside functions. Loaded when program starts, can be modified
Code: loaded when prog starts, doesn't change
$fp
Saved argument registers
Saved return address
Saved saved registers
Local arrays and structures
$sp
$sp
Stack
Dynamic data
$gp
Static data
Text
pc
Reserved
1000 0000 hex
0040 0000 hex
0 hex
6 Great Ideas in Computer Architecture
1. Abstraction (Layers of Representation/Interpretation)
2. Moore's Law
Designing through trends
3. Principle of Locality
Memory Hierarchy
4. Parallelism
5. Performance Measurement and Improvement
6. Dependability via Redundancy
Two's-Complement Representation
- treats 0 as positive
- 32-bit word represents 2^{32} numbers from -2^{31} to $2^{31}-1$
Components of a Computer
Processor
Control
Datapath
PC
Registers
Arithmetic & Logic Unit (ALU)
Address
Write Data
Read Data
Memory
Input
Program
Bytes
Data
Output
Processor-Memory Interface
I/O-Memory Interfaces
Stack
- free when function returns
- Last in first out
Common Memory Problems
- Using uninitialized values
- Using memory that you don't own
- Improper use of free/malloc by messing with the pointer handle returned by malloc/calloc
- Memory leaks
R format: used for instructions with three registers. lw and sw (and branches (beq and bne))
Opcode rs rt rd shamt funct
opcode = 0
I format: 16-bit field only represents number up to 32,767
If instruction has immediate, then it use at most 2 registers
used for lw, sw, beq, bne, branches, and with immediate
Decoding with larger immediate
- Load Upper Imm
addiu $t0, $t0, 0xABCD
lui $at, 0xABCD
ori $t0, $at, 0xCDEF
addu $t0, $t0, $t1
fw.c -> cpp -> fw.i -> compiler
- cpp replaces comments with a single space
- cpp commands begins with "#"
C standard Type
(unsigned) char / signed char 1
(unsigned) int 4
(unsigned) short 2
(unsigned) long 4
float 4 byte
double 8 byte
long long 32 8
Levels of Representation/Interpretation
High Level Language
Assembly Language
Machine Language
Machine Interpretation
Hardware Architecture Description
Architecture Implementation
Logic Circuit Description

SIFT REFERENCE GUIDE (V.1.1) – CREATING TIMELINES WITH THE SIFT WORKSTATION

SANS COMPUTER FORENSICS and INCIDENT RESPONSE

THE PURPOSE OF THIS REFERENCE GUIDE IS TO WALK THROUGH THE PROCESS OF BOOTING THE SIFT WORKSTATION, CREATING A TIMELINE ("SUPER" OR "MICRO") AND REVIEWING IT.

1. VISIT: http://computer-forensics11.sans.org/community/downloads

1 → **load:** SIFT Workstation VM Appliance

Do SIFT Workstation Installation

2 → **2. BOOT SIFT VM**

Login: *sansforensics*
Password: *forensics*

3 → $ sudo su

3. ELEVATE PRIVS

4 → **4. CONNECT IMAGE TO SIFT**

Plug hard drive to physical host and attach to SIFT VM

5

5. HARD DRIVE MOUNTING *(if you are using log2timeline-sift and Single DD you can skip to 7-A)*

EWF/

SINGLE OR SPLIT IMAGE (2 options):

`nt_ewf.py image.E01 /mnt/ewf`

`# image.E01 /mnt/ewf/`

Not Needed For 7-A

`# mount -t ntfs -o ro,loop,show_sys_files,streams_interface=wind offset=#### /mnt/ewf/<image> /mnt/windows_mount/`

MOUNT TO MOUNT POINT

DD

SINGLE IM

`# mount -t ntfs -o w_sys_files,streams_interface=windows,offset=#### image.dd /m ws_mount/`

SPLIT IMAGE (2 step proc

`# affuse image.001 /mnt/aff`

`# mount –t ntfs-3g –o loop,ro,show mnt/aff/<image> /mnt/windows`

6

HOW TO CALCULATE THE OFFSET FOR MOUNTING

1. Run mmls to query partition layout
 # mmls image.E01
2. Identify partition and byte offset
3. (Partition byte offset) x (bytes per sector) = **offset #### to use!**
Example: 63 X 512 = 32256

Note: If needed, repeat for each partition. Make new mount point:
 # mkdir /mnt/windows_mount2/

6. log2timeline default timezone is set to examiner local host. To change use -z [TIMEZONE] option. To list all available timezones:
 # log2timeline -z list

7-A & 7-B

8

7-A: AUTOMATED SUPER TIMELINE CREATION

log2timeline-sift -o –z [TIMEZONE] -p [PARTITION #] -i [IMAGE FILE]

DISK IMAGE (prompt for partition, mount, and run):

XP `# log2timeline-sift –z EST5EDT -i image`

WIN7 `# log2timeline-sift -win7 -z EST5EDT -i image`

FOR PARTITION (mount and run using all applicable pl

XP `# log2timeline-sift –z EST5EDT -p 0 -i parti`

WIN7 `# log2timeline-sift -win7 -z EST5EDT tion`

OTHER USAGE EXAMPLES:

Display list of available pl
 # log2timeline -f list
Run log2timeline use use only specific plugins:
 # log2timeline-sif prefetch –z EST5EDT -i image.dd
Help (man pag
 # log2timeli

7-B: MANUAL "MICRO" TIMELINE CREATION

`TIONS] [-f FORMAT] [-z TIMEZONE] [-o OUTPUT MODULE] [-w G_FILE/LOG_DIR [--] [FORMAT FILE OPTIONS]`

TEM METADATA (using log2timeline or fls)

stem data w/log2timeline from mounted file system:
 # log mft -o mactime –r -z EST5EDT -w
mft.bo lume/
OR Extract M age using Sleuthkit:
 # fls -m "" -o offs .dd > fls.body
Convert body file for format w/ mactime:
 # mactime –b fls.body –d

ARTIFACTS (run l2l on mounted file s h plugins recursively)

Extract artifacts w/ log2timeline and ru d file system:
 # log2timeline -f firefox3,chrome -o mactim EDT -w
web.body /mnt/volume/
Convert body file format to CSV format w/ mactime:
 # mactime –b log2timeline.body –d > log2timeline.csv

HELP? OPTIONS? USAGE?
log2timeline -help
Log2timeline-sift -help
L2t_process -help

OTHER log2timeline OUTPUT FORMATS
Note: CSV is Default Output
-BeeDocs - Mac OS X visualization tool
-CEF - Common Event Format - ArcSight
-CFTL - XML file- CyberForensics TimeLab visualization tool
-CSV - comma separated value file
-Mactime - Both older and newer version of the format supported for use by TSK's mactime
-SIMILE - XML file - SIMILE timeline visualization widget
-SQLite - SQLite database
-TLN - Tab Delimited File
-TLN - Format used by some of H Carvey tools, expressed as a ASCII output
-TLNX - Format used by some of H Carvey tools, expressed as a XML document

8. CSV F PUT ***(/cases/timeline-output-folder)***

-dat e event, in the format of MM/DD/YYYY
-t of day, expressed in a 24h format, HH:MM:SS
-z: the timezone that was used to call the tool with.
-m: MACB meaning of the fields, comp w/ mactime format.
-source: Source short name (i.e. registry entries are REG)
-sourcetype: Desc of the source ("Internet Explorer" instead of WEBHIST)
-type: Timestamp type (i.e. "Last Accessed", "Last Written")
-user: Username associated with the entry, if one is available.
-host: Hostname associated with the entry, if one is available.
-short: Contains less text than the full description field.
-desc: where majority info is stored, the actual parsed desc of the entry.
-version: Version number of the timestamp object.
-filename: Filename with the full path that contained the entry
-inode: inode number of the file being parsed.
-notes: Some input modules insert additional information in the form of a note, which comes here. Or it can be used during the review.
-format: Input module name used to parse the file.
-extra: Additional information parsed is joined together and put here.

9

9. FILTER TIMELINE

Filter timeline with date range to include only:
l2t_process -b timeline.csv MM-DD-YYYY..MM-DD-YYYY > filtered.csv
Filter timeline with keyword list (one term per line in keywords.txt):
l2t_process -b timeline.csv -k keywords.txt > filtered.csv
What sources are in your timeline?
awk–F , '{print $6;}' timeline.csv| grep–v sourcetype|sort | uniq
Find all LNK files that reference E Drive
grep"Shortcut LNK" timeline.csv| grep"E:"
FiindMountPoints2 entries that reference E Drive
grep"MountPoints2 key" timeline.csv} | grep"E drive"
grepUSB timeline.csv| grep"SetupAPILog"

File System	M	A	C	B
Ext2/3	Modified	Accessed	Changed	N/A
FAT	Written	Accessed	N/A	Created
NTFS	File Modified	Accessed	MFT Modified	Created
UFS	Modified	Accessed	Changed	N/A

10

10. CONNECT TO SIFT

✓ > SETTINGS -> OPTIONS -> Shared Folders -> Always Enabled (Check)

✓ 2. SIFT Desktop > VMware-Shared-Drive

✓ Access from a Win Machine \\SIFTWORKSTATION

11

11. REVIEW TIMELINE

Review timelines using:
- Open, Soft, Filter with Excel
- Import into SPLUNK
- SIMILE
- Tapestry

log2timeline PARSING PLUGINS

apache2_error - Apache2 error log file
chrome - Chrome history file
encase_dirlisting - CSV file that is exported from encase
evt - Windows 2k/XP/2k3 Event Log
evtx - Windows Event Log File (EVTX)
exif - Metadata information from files using ExifTool
ff_bookmark - Firefox bookmark file
firefox2 - Firefox 2 browser history
firefox3 - Firefox 3 history file
ftk_dirlisting - CSV file that is exported from FTK Imager (dirlisting)
generic_linux - Generic Linux logs that start with MMM DD HH:MM:SS
iehistory - index.dat file containg IE history
iis - IIS W3C log file
isatxt - ISA text export log file
jp_ntfs_change - CSV output file from JP (NTFS Change log)
mactime - Body file in the mactime format
mcafee - Log file
mft - NTFS MFT file
mssql_errlog - ERRORLOG file produced by MS SQL server
ntuser - NTUSER.DAT registry file
opera - Opera's global history file
oxml - OpenXML document pcap
pcap - PCAP file
pdf - Available PDF document metadata
prefetch - Prefetch directory
recycler - Recycle bin directory
restore 0.9 - Restore point directory
safari - Safari History.plist file
sam - SAM registry file
security - SECURITY registry file
setupapi - SetupAPI log file in Windows XP
skype_sql - Skype database
software - SOFTWARE registry file
sol - .sol (LSO) or a Flash cookie file
squid - Squid access log (http_emulate off)
syslog - Linux Syslog log file
system - SYSTEM registry file
tln - Body file in the TLN format
volatility - Volatility output files (psscan2, sockscan2, ...)
win_link - Windows shortcut file (or a link file)
wmiprov - wmiprov log file
xpfirewall - XP Firewall log

List plugins # log2timeline -f list
...HELP EXPAND THIS LIST. BUILD PLUGINS!!!

BY DAVID NIDES (12/16/2011)
TWITTER: @DAVNADS
BLOG: DAVNADS.BLOGSPOT.COM
EMAIL: DNIDES@KPMG.COM

CREDITS TO: ED GOINGS, ROB LE KRISTINN GUDJONSSON, KPMG &

QUESTIONS/FEEDBACK–CONTACT US!

KEY
Red text – image/source
Blue text – mount point
Purple text – output file
Green text – log2timeline plugins
Brown text - TimeZone

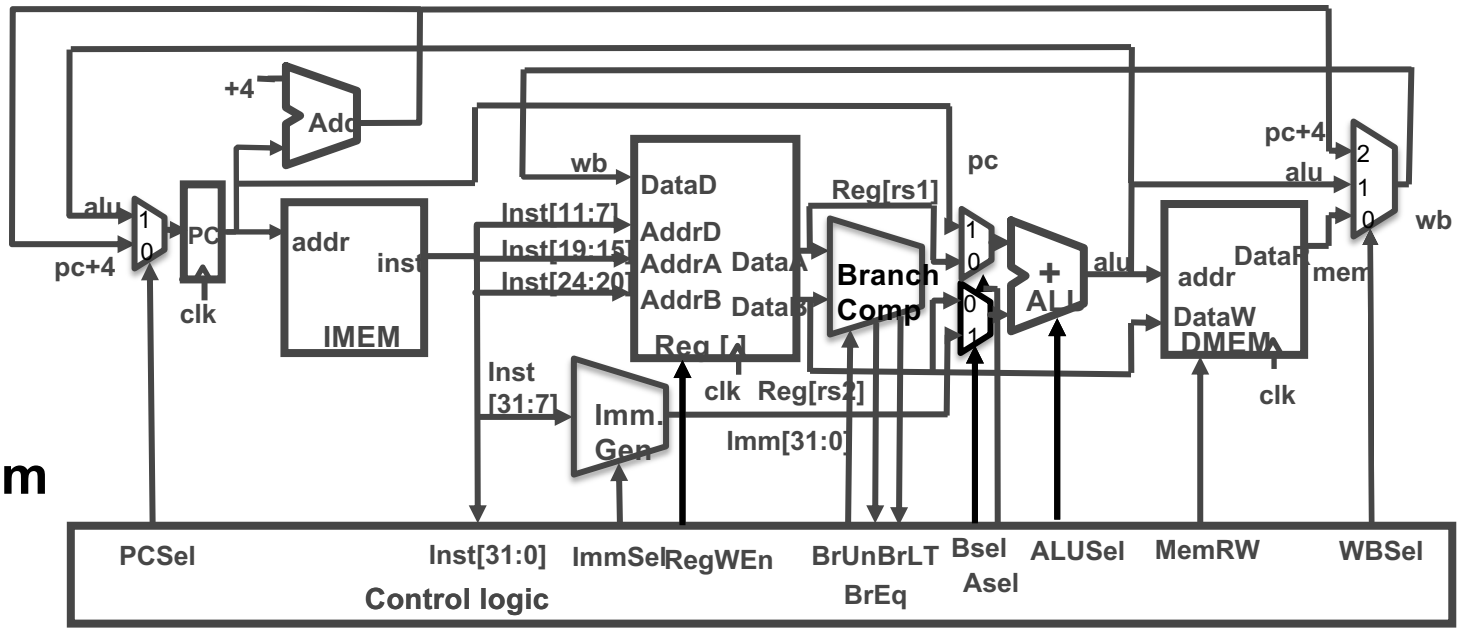
Content

- Main topics
  - Number representation
  - C
  - RISC-V
- Plus general ”Computer Architecture” knowledge
- Everything till lecture 8 CALL – including lecture 8

Peer Instruction: Critical Path

Critical path for addi

R[rd] = R[rs1]+imm



A. $t_{clk\text{-}q} + t_{Add} + t_{IMEM} + t_{Reg} + t_{BComp} + t_{ALU} + t_{DMEM} + t_{mux} + t_{Setup}$

B. $t_{clk\text{-}q} + t_{IMEM} + \max\{t_{Reg}, t_{Imm}\} + t_{ALU} + 2t_{mux} + t_{Setup}$

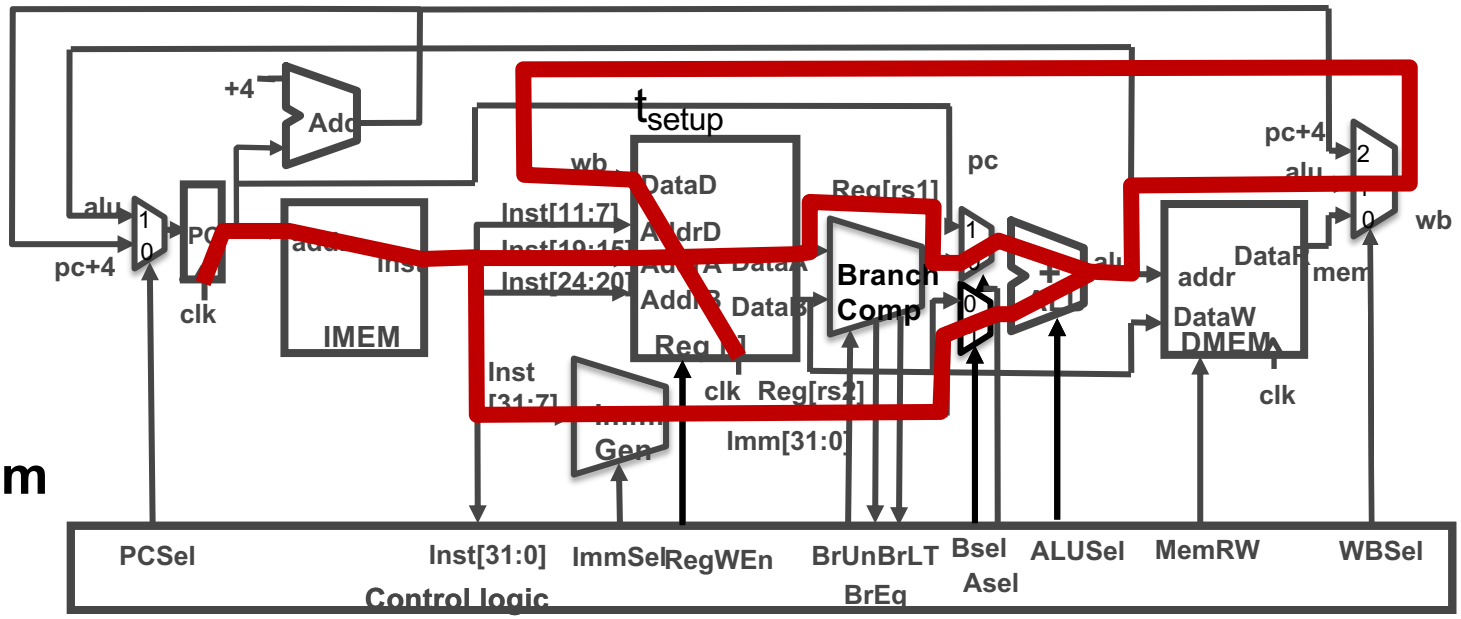
C. $t_{clk\text{-}q} + t_{IMEM} + \max\{t_{Reg}, t_{Imm}\} + t_{ALU} + 3t_{mux} + t_{DMEM} + t_{Setup}$

D. None of the above

Peer Instruction: Critical Path

Critical path for addi

R[rd] = R[rs1]+imm



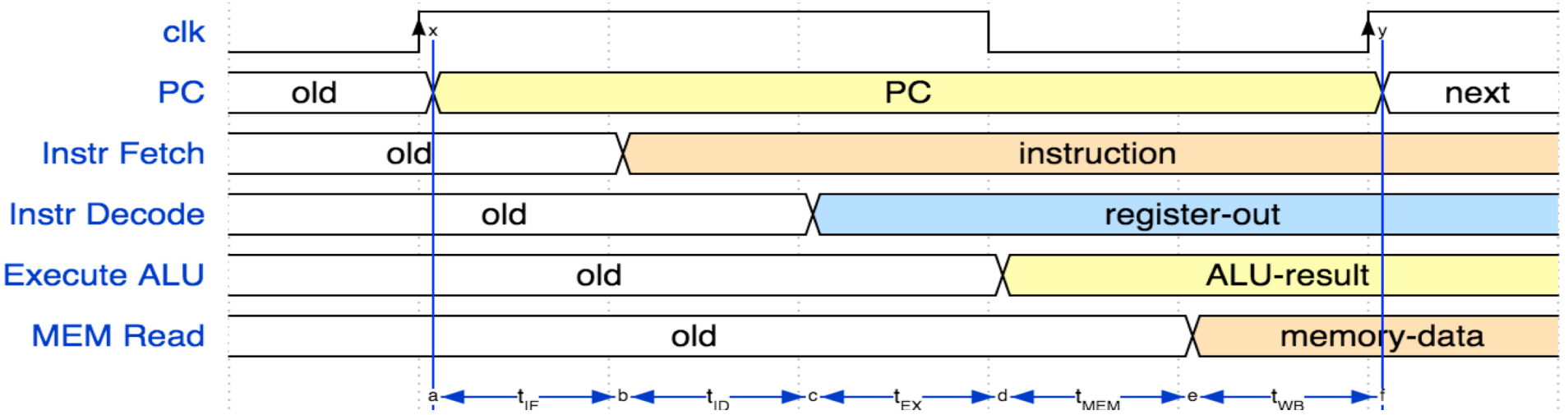
A. $t_{clk\text{-}q} + t_{Add} + t_{IMEM} + t_{Reg} + t_{BComp} + t_{ALU} + t_{DMEM} + t_{mux} + t_{Setup}$

B. $t_{clk\text{-}q} + t_{IMEM} + \max\{t_{Reg}, t_{Imm}\} + t_{ALU} + 2*t_{mux} + t_{Setup}$

C. $t_{clk\text{-}q} + t_{IMEM} + \max\{t_{Reg}, t_{Imm}\} + t_{ALU} + 3*t_{mux} + t_{DMEM} + t_{Setup}$

D. None of the above

Instruction Timing



IF	ID	EX	MEM	WB	Total
I-MEM	Reg Read	ALU	D-MEM	Reg W	
200 ps	100 ps	200 ps	200 ps	100 ps	**800 ps**

Instruction Timing

Instr	IF = 200ps	ID = 100ps	ALU = 200ps	MEM=200ps	WB = 100ps	Total
add	X	X	X		X	600ps
beq	X	X	X			500ps
jal	X	X	X		X	600ps
lw	X	X	X	X	X	800ps
sw	X	X	X	X		700ps

- Maximum clock frequency
  - f_{max} = 1/800ps = 1.25 GHz
- Most blocks idle most of the time
  - E.g. $f_{max,ALU}$ = 1/200ps = 5 GHz!

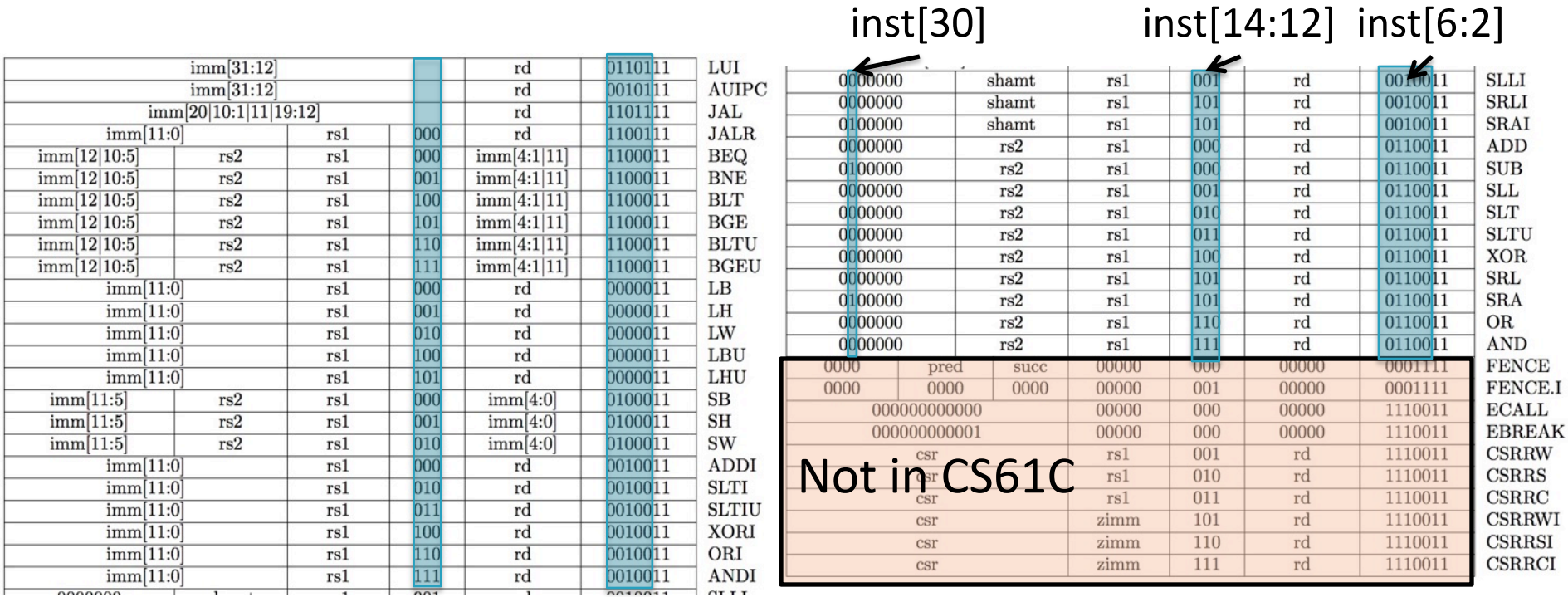
Control Logic Truth Table

Inst[31:0]	BrEq	BrLT	PCSel	ImmSel	BrUn	ASel	BSel	ALUSel	MemRW	RegWEn	WBSel
`add`	*	*	+4	*	*	Reg	Reg	Add	Read	1	ALU
`sub`	*	*	+4	*	*	Reg	Reg	Sub	Read	1	ALU
(R-R Op)	*	*	+4	*	*	Reg	Reg	*(Op)*	Read	1	ALU
`addi`	*	*	+4	I	*	Reg	Imm	Add	Read	1	ALU
`lw`	*	*	+4	I	*	Reg	Imm	Add	Read	1	Mem
`sw`	*	*	+4	S	*	Reg	Imm	Add	Write	0	*
`beq`	0	*	+4	B	*	PC	Imm	Add	Read	0	*
`beq`	1	*	ALU	B	*	PC	Imm	Add	Read	0	*
`bne`	0	*	ALU	B	*	PC	Imm	Add	Read	0	*
`bne`	1	*	+4	B	*	PC	Imm	Add	Read	0	*
`blt`	*	1	ALU	B	0	PC	Imm	Add	Read	0	*
`bltu`	*	1	ALU	B	1	PC	Imm	Add	Read	0	*
`jalr`	*	*	ALU	I	*	Reg	Imm	Add	Read	1	PC+4
`jal`	*	*	ALU	J	*	PC	Imm	Add	Read	1	PC+4
`auipc`	*	*	+4	U	*	PC	Imm	Add	Read	1	ALU

Control Realization Options

- ROM
  - "Read-Only Memory"
  - Regular structure
  - Can be easily reprogrammed
    - fix errors
    - add instructions
  - Popular when designing control logic manually
- Combinatorial Logic
  - Today, chip designers use logic synthesis tools to convert truth tables to networks of gates

RV32I, a nine-bit ISA!



imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
0000	pred	succ	00000	000	00000	0001111	FENCE
0000	0000	0000	00000	001	00000	0001111	FENCE.I
000000000000			00000	000	00000	1110011	ECALL
000000000001			00000	000	00000	1110011	EBREAK
csr			rs1	001	rd	1110011	CSRRW
csr			rs1	010	rd	1110011	CSRRS
csr			rs1	011	rd	1110011	CSRRC
csr			zimm	101	rd	1110011	CSRRWI
csr			zimm	110	rd	1110011	CSRRSI
csr			zimm	111	rd	1110011	CSRRCI

Instruction type encoded using only 9 bits
inst[30],inst[14:12], inst[6:2]

Combinational Logic Control

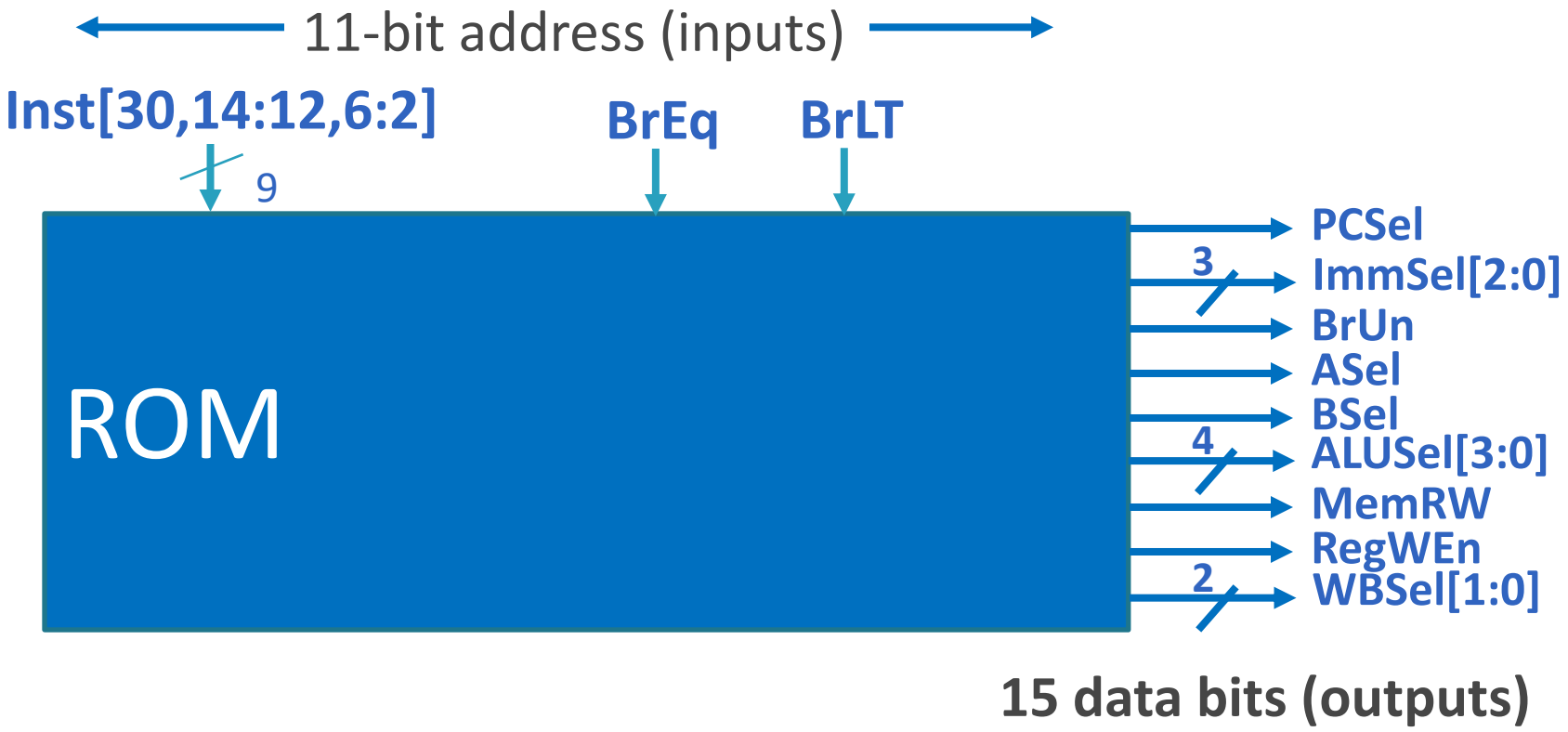
- Simplest example: BrUn

inst[14:12] inst[6:2] = Branch

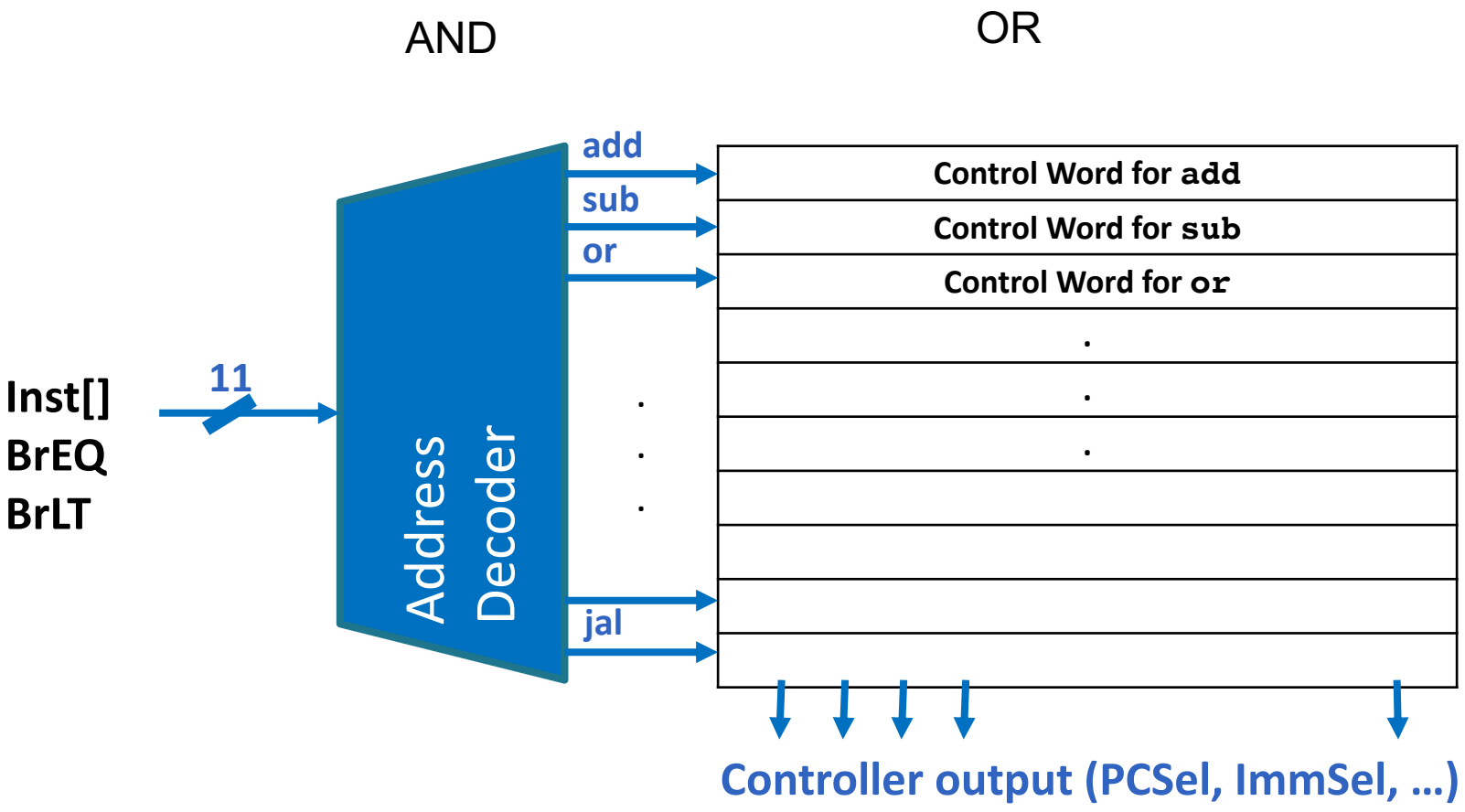
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU

- **How to decode whether BrUn is 1?**
  - **BrUn = Inst [13] • Branch**

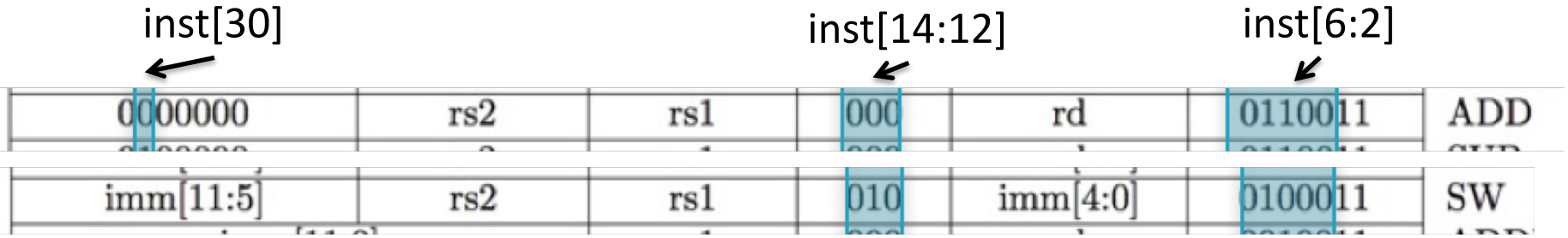
ROM-based Control



ROM Controller Implementation



Controller: AND logic (add, sw)



- add= $\overline{inst[2]} \bullet \overline{inst[3]} \bullet inst[4] \bullet inst[5] \bullet \overline{inst[6]} \bullet \overline{inst[12]} \bullet \overline{inst[13]} \bullet \overline{inst[14]} \bullet \overline{inst[30]}$
- sw= $\overline{inst[2]} \bullet \overline{inst[3]} \bullet \overline{inst[4]} \bullet inst[5] \bullet \overline{inst[6]} \bullet \overline{inst[12]} \bullet inst[13] \bullet \overline{inst[14]}$

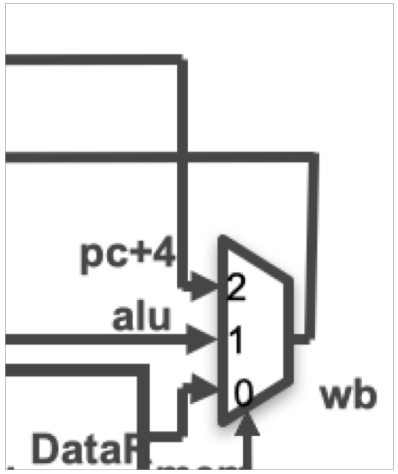
AND Controller Logic

		2		3		4		5		6		12		13		14		30		unused	
add	xor(i[2],	1	) • xor(i[3],	1	) • xor(i[4],	0	) • xor(i[5],	0	) • xor(i[6],	1	) • xor(i[12],	1	) • xor(i[13],	1	) • xor(i[14],	1	) • (xor(i[30],	1	) +	0	)
sw	xor(i[2],	1	) • xor(i[3],	1	) • xor(i[4],	1	) • xor(i[5],	0	) • xor(i[6],	1	) • xor(i[12],	1	) • xor(i[13],	1	) • xor(i[14],	1	) • (xor(i[30],	1	) +	1	)
...																					

- bne_t = bne • $\overline{\textbf{BrEQ}}$
- bne_f = bne • **BrEQ**

Controller: OR logic

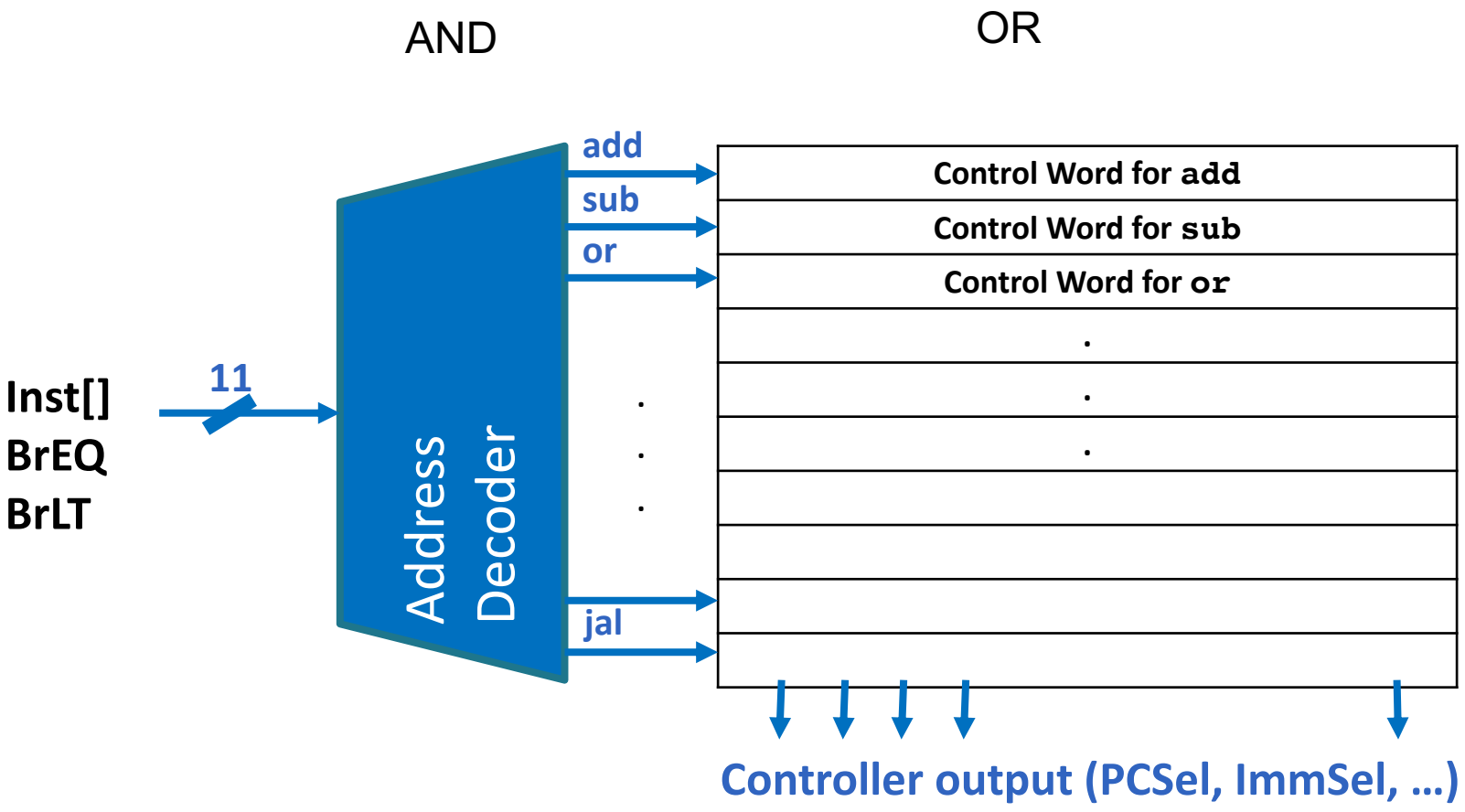
- MemRW = sw + sh + sb
- WBSel[0] = add + (all reg. to reg.) + AUIPC + …
- WBSel[1] = JALR + JAL
- PCSel = beq_t + bne_t + blt_t + bltu_t + jal + jalr
- …



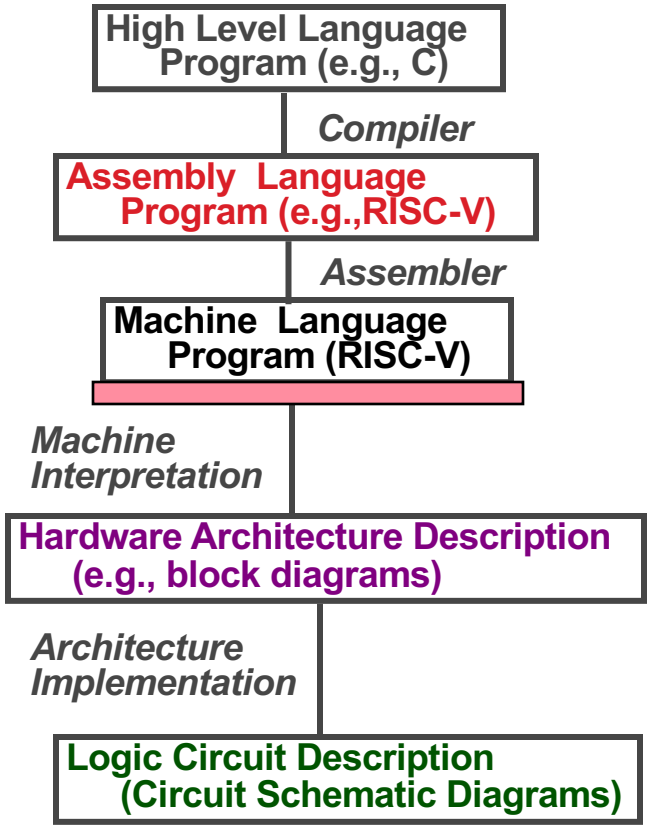
Control Logic Truth Table

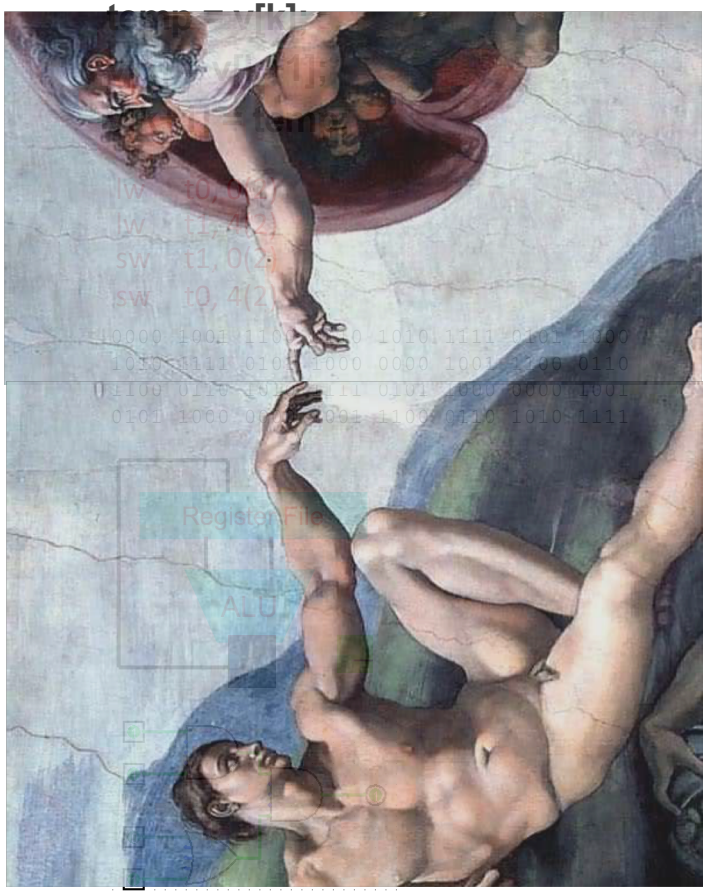
Inst[31:0]	BrEq	BrLT	PCSel	ImmSel	BrUn	ASel	BSel	ALUSel	MemRW	RegWEn	WBSel
`add`	*	*	+4	*	*	Reg	Reg	Add	Read	1	ALU
`sub`	*	*	+4	*	*	Reg	Reg	Sub	Read	1	ALU
(R-R Op)	*	*	+4	*	*	Reg	Reg	*(Op)*	Read	1	ALU
`addi`	*	*	+4	I	*	Reg	Imm	Add	Read	1	ALU
`lw`	*	*	+4	I	*	Reg	Imm	Add	Read	1	Mem
`sw`	*	*	+4	S	*	Reg	Imm	Add	Write	0	*
`beq`	0	*	+4	B	*	PC	Imm	Add	Read	0	*
`beq`	1	*	ALU	B	*	PC	Imm	Add	Read	0	*
`bne`	0	*	ALU	B	*	PC	Imm	Add	Read	0	*
`bne`	1	*	+4	B	*	PC	Imm	Add	Read	0	*
`blt`	*	1	ALU	B	0	PC	Imm	Add	Read	0	*
`bltu`	*	1	ALU	B	1	PC	Imm	Add	Read	0	*
`jalr`	*	*	ALU	I	*	Reg	Imm	Add	Read	1	PC+4
`jal`	*	*	ALU	J	*	PC	Imm	Add	Read	1	PC+4
`auipc`	*	*	+4	U	*	PC	Imm	Add	Read	1	ALU

ROM Controller Implementation



Call home, we've made HW/SW contact!





“And In conclusion…”

- We have built a processor!
  - Capable of executing all RISC-V instructions in one cycle each
  - Not all units (hardware) used by all instructions
  - Critical path changes
- 5 Phases of execution
  - IF, ID, EX, MEM, WB
  - Not all instructions are active in all phases
- Controller specifies how to execute instructions
  - Implemented as ROM or logic