CSCI E-93 Computer Architecture

MIN16 Computer Architecture

Final Project Presentation

Masakazu Tanami

Jan 25, 2017

Overview

- 16-bit RISC CPU architecture
- 8-bits Byte and 16-bits Word memory access
- 16 registers including program counter and ALU flags
- 4 Addressing Modes (R-type, I-type, J-type, O-type)
- Arithmetic, Logical, Shift Operations (Signed/Unsigned/Immediate)
- Jump, Branch Flow Control Operations
- Load, Store Memory Access Operations
- 46 Instruction Mnemonics

Register File

NUMBER	NAME	USE	SAVED
0	r0	The constant value 0	N/A
1	at	Asembler Temporary	No
2	sp	Stack Pointer	Yes
3	fp	Frame Pointer	Yes
4	ra	Return Address	Yes
5	rb	General Use	No
6	rc	General Use	No
7	rd	General Use	No
8	s0	Saved Temporary	Yes
9	s1	Saved Temporary	Yes
10	t0	Temporary	No
11	t1	Temporary	No
12	hi	High Part	No
13	lo	Low Part	No

Status Flag Special Register

BIT#	NAME	DESCRIPTION	CATEGORY
0	ZF	Zero Flag	Status
1	SF	Sign Flag	Status
2	CF	Carry Flag	Status
3	OF	Overflow Flag	Status
4	EQ	Equal Flag	Status

Operation Code

R-type

I-type

J-type

O-type

OPCODE	HEX	INSTRUCTIONS
0000	0x0	Signed Arithmetic
0001	0x1	Unsigned Arithmetic
0010	0x2	Logical
0011	0x3	Shift
0100	0x4	Immediate Signed Arithmetic
0101	0x5	Immediate Unsigned Arithmetic
0110	0x6	Immediate Logical
0111	0x7	Immediate Shift
1000	0x8	Reserved
1001	0x9	Reserved
1010	0xA	Jump
1011	0xB	Branch
1100	0xC	Memory Load and Store
1101	0xD	Move HI/LO Data
1110	0xE	Reserved
1111	0xF	Reserved

Addressing Modes

- R-Type (Two Registers)

```
[op] [func]   [rd]  [rs] [sbz]
 0000   00     0000  0000  00
```

- J-Type (Jump to Target Address)

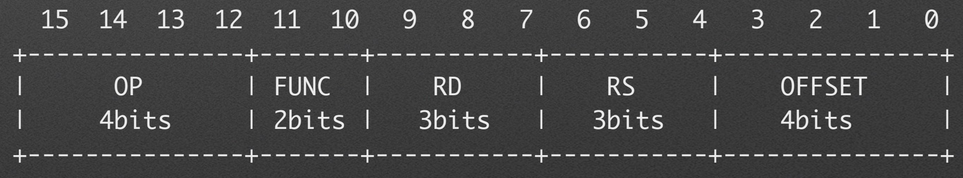
```
[op] [func]  [target]
1000   00    0000000000
```

- I-Type (One Register, One Immediate)

```
[op] [func]  [rd] [imm]
0100   00    0000 000000
```

- O-Type (Two Registers, One Offset)

```
[op] [func]  [rd] [rs] [offset]
 1000   00    000  000    0000
```



Register File

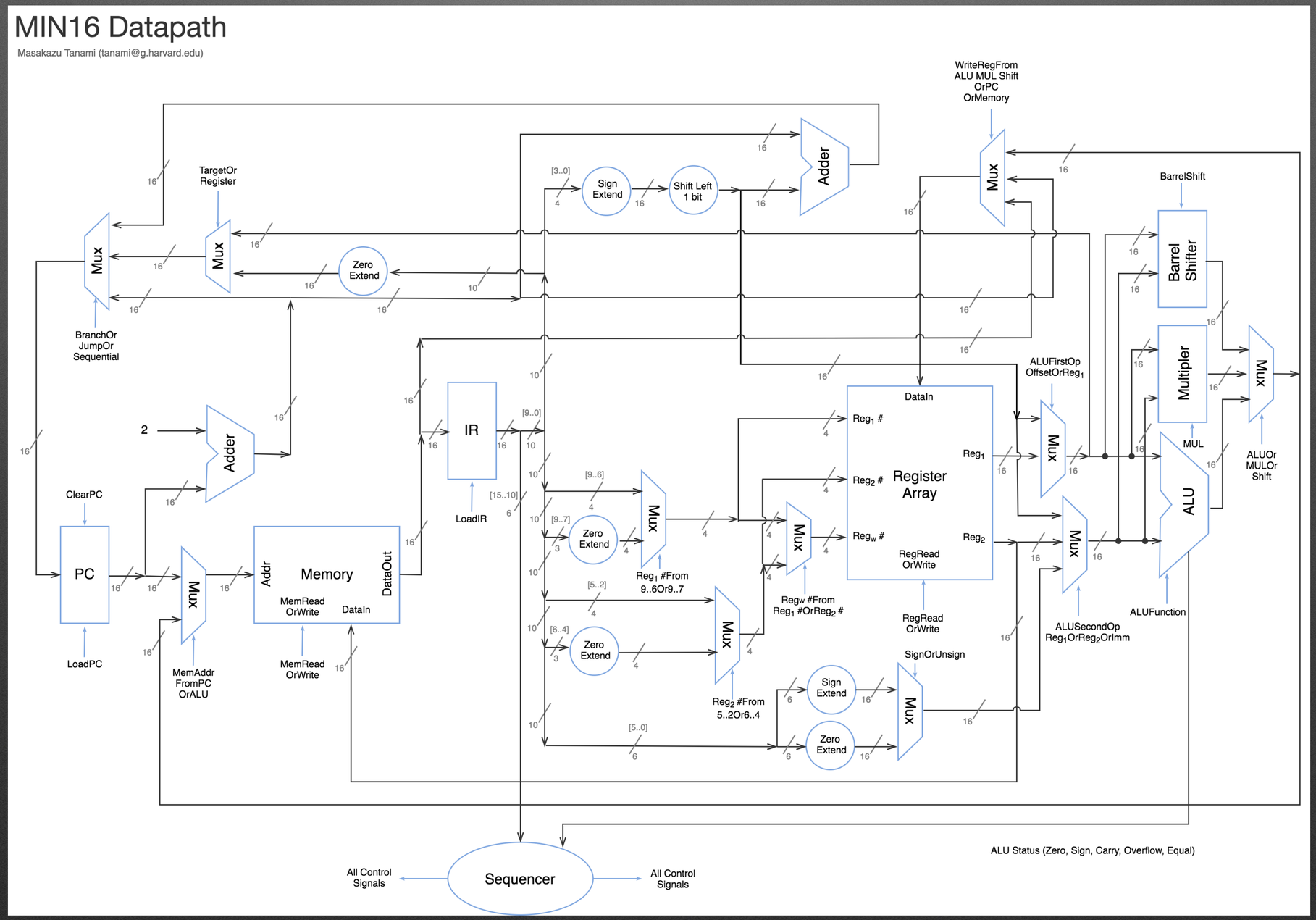
NUMBER	NAME	USE	SAVED
0	r0	The constant value 0	N/A
1	at	Asembler Temporary	No
2	sp	Stack Pointer	Yes
3	fp	Frame Pointer	Yes
4	ra	Return Address	Yes
5	rb	General Use	No
6	rc	General Use	No
7	rd	General Use	No
8	s0	Saved Temporary	Yes
9	s1	Saved Temporary	Yes
10	t0	Temporary	No
11	t1	Temporary	No
12	hi	High Part	No
13	lo	Low Part	No

Instruction Mnemonic Map

Mode	op \ func	0b00	0b01	0b10	0b11	
R-type	0x0	ADD	SUB	MUL	SLT	Signed Arithmetic
	0x1	ADDU	SUBU	MULU	SLTU	Unsigned Arithmetic
	0x2	AND	OR	XOR	NOR	Logical
	0x3	SLL	SRL	SRA	ROTL	Shift
I-type	0x4	ADDI	SUBI	MULI	SLTI	Immediate Signed Arithmetic
	0x5	ADDIU	SUBIU	MULIU	SLTIU	Immediate Unsigned Arithmetic
	0x6	ANDI	ORI	XORI	NORI	Immediate Logical
	0x7	SLLI	SRLI	SLAI	ROTLI	Immediate Shift
	0x8					Reserved
	0x9					Reserved
J-type	0xA	J	JAL	JR	JALR	Jump
O-type	0xB	BEQ	BNE			Branch
	0xC	LW	LB	SW	SB	Memory Load and Store
	0xD	MFHI	MFLO	MTHI	MTLO	Move HI/LO Data
	0xE					Reserved
	0xF					Reserved

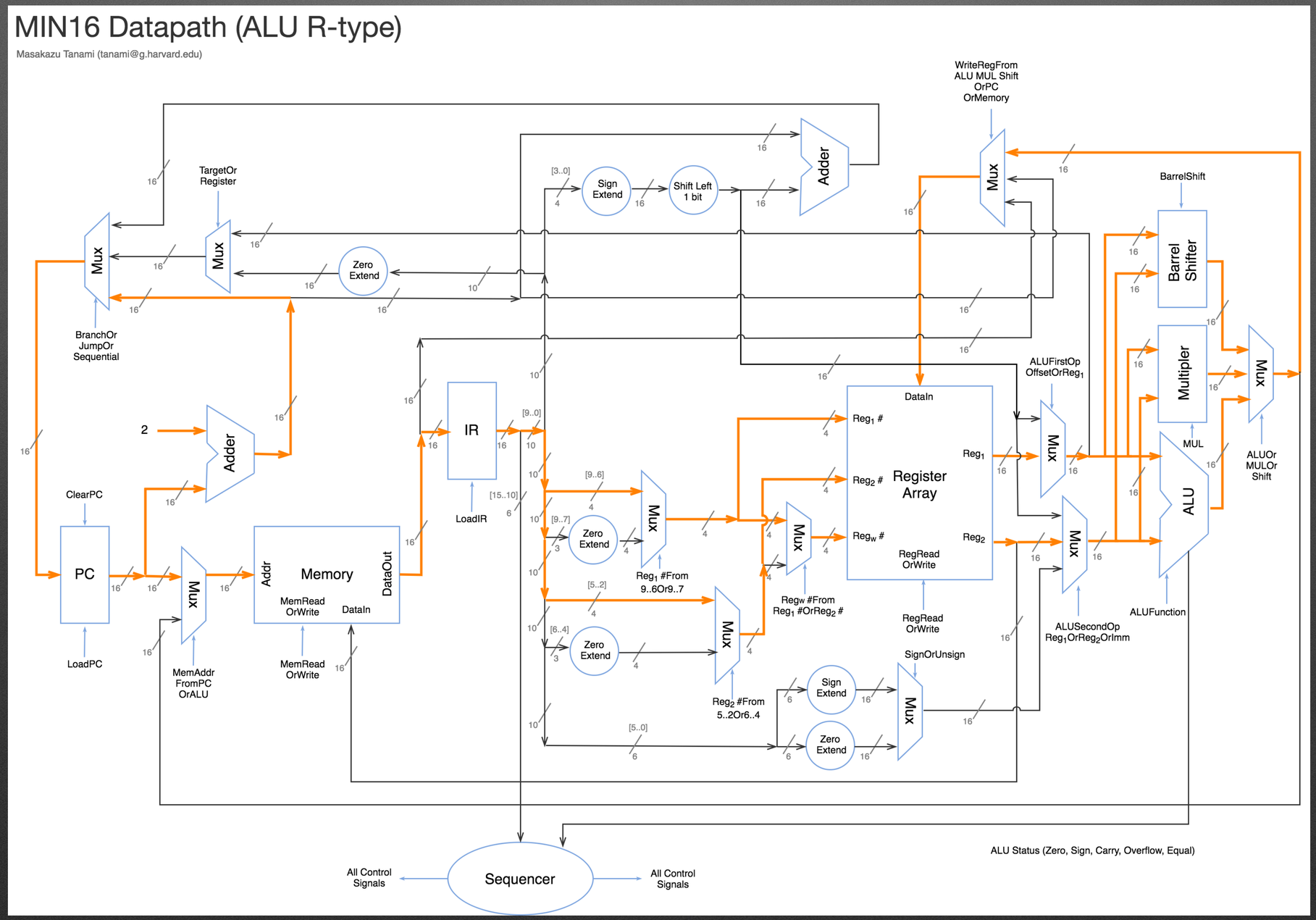
CSCI E-93 Computer Architecture

Reg_1 =>dest, Reg_2 =>source



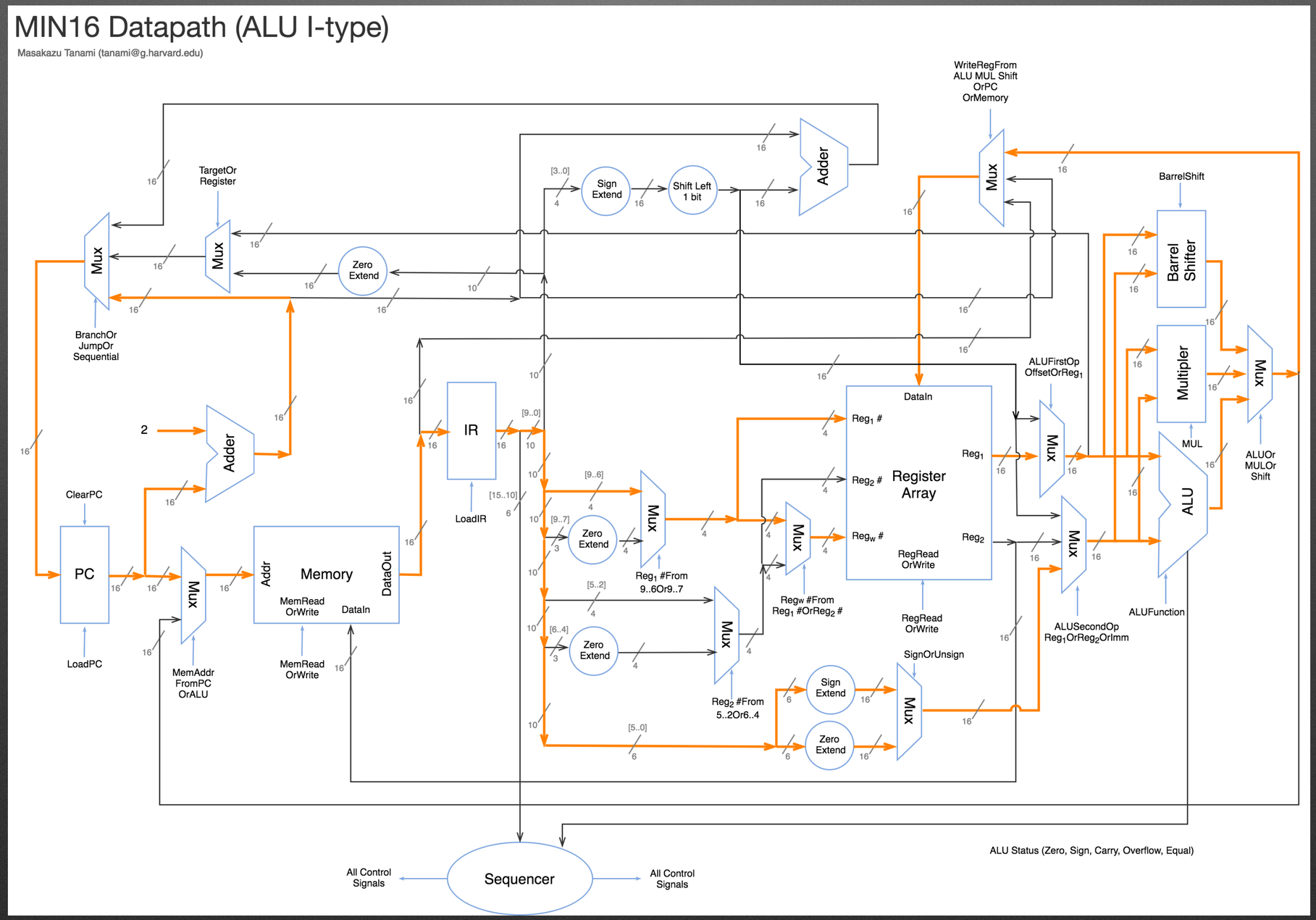
CSCI E-93 Computer Architecture

ADD $rd, $rs

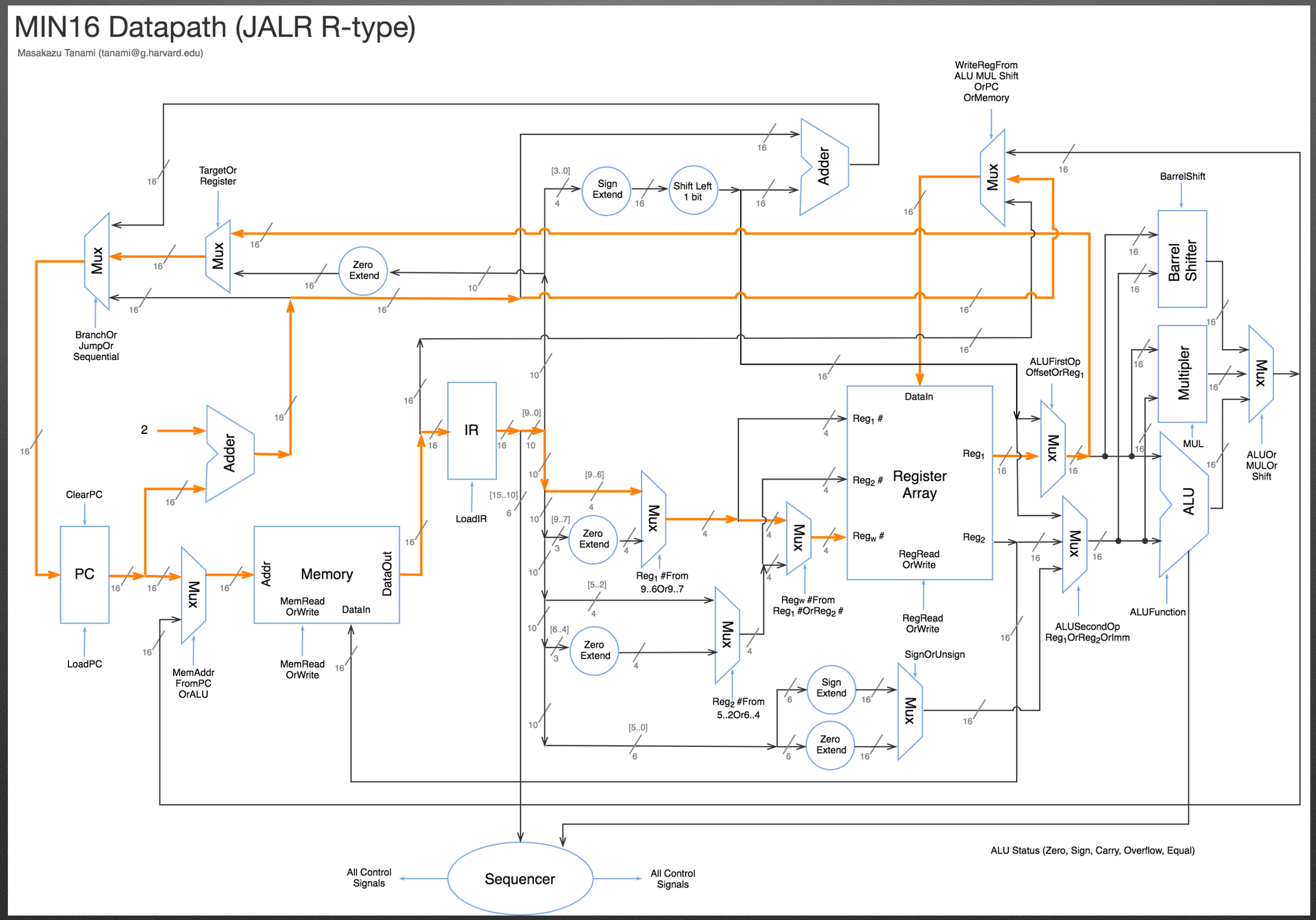


CSCI E-93 Computer Architecture

ANDI $rd, 0

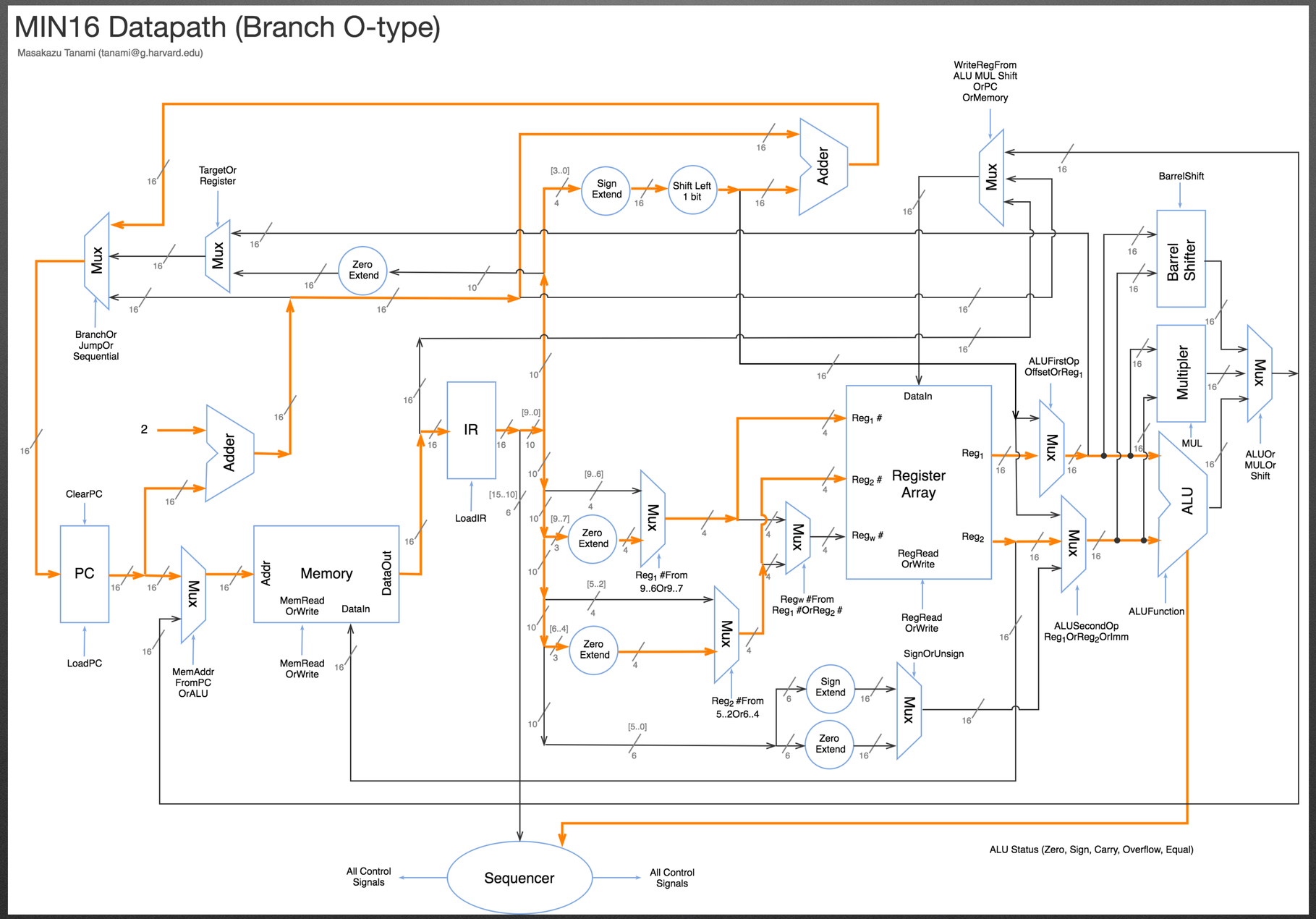


JALR $rd


MIN16 Datapath (JALR R-type)
Masakazu Tanami (tanami@g.harvard.edu)
WriteRegFrom
ALU MUL Shift
OrPC
OrMemory
TargetOr
Register
BranchOr
JumpOr
Sequential
Sign
Extend
Shift Left
1 bit
Adder
Zero
Extend
Mux
BarrelShift
Barrel
Shifter
Multipler
MUL
ALUOr
MULOr
Shift
ALU
ALUFirstOp
OffsetOrReg1
DataIn
Reg1 #
Reg2 #
Regw #
Register
Array
Reg1
Reg2
RegRead
OrWrite
ALUSecondOp
Reg1OrReg2OrImm
ALUFunction
SignOrUnsign
IR
LoadIR
2
ClearPC
PC
LoadPC
MemAddr
FromPC
OrALU
Memory
Addr
DataOut
MemRead
OrWrite
DataIn
Reg1 #From
9..6Or9..7
Regw #From
Reg1 #OrReg2 #
Reg2 #From
5..2Or6..4
[3..0]
[9..0]
[9..6]
[9..7]
[15..10]
[5..2]
[6..4]
[5..0]
Sequencer
All Control
Signals
All Control
Signals
ALU Status (Zero, Sign, Carry, Overflow, Equal)

CSCI E-93 Computer Architecture

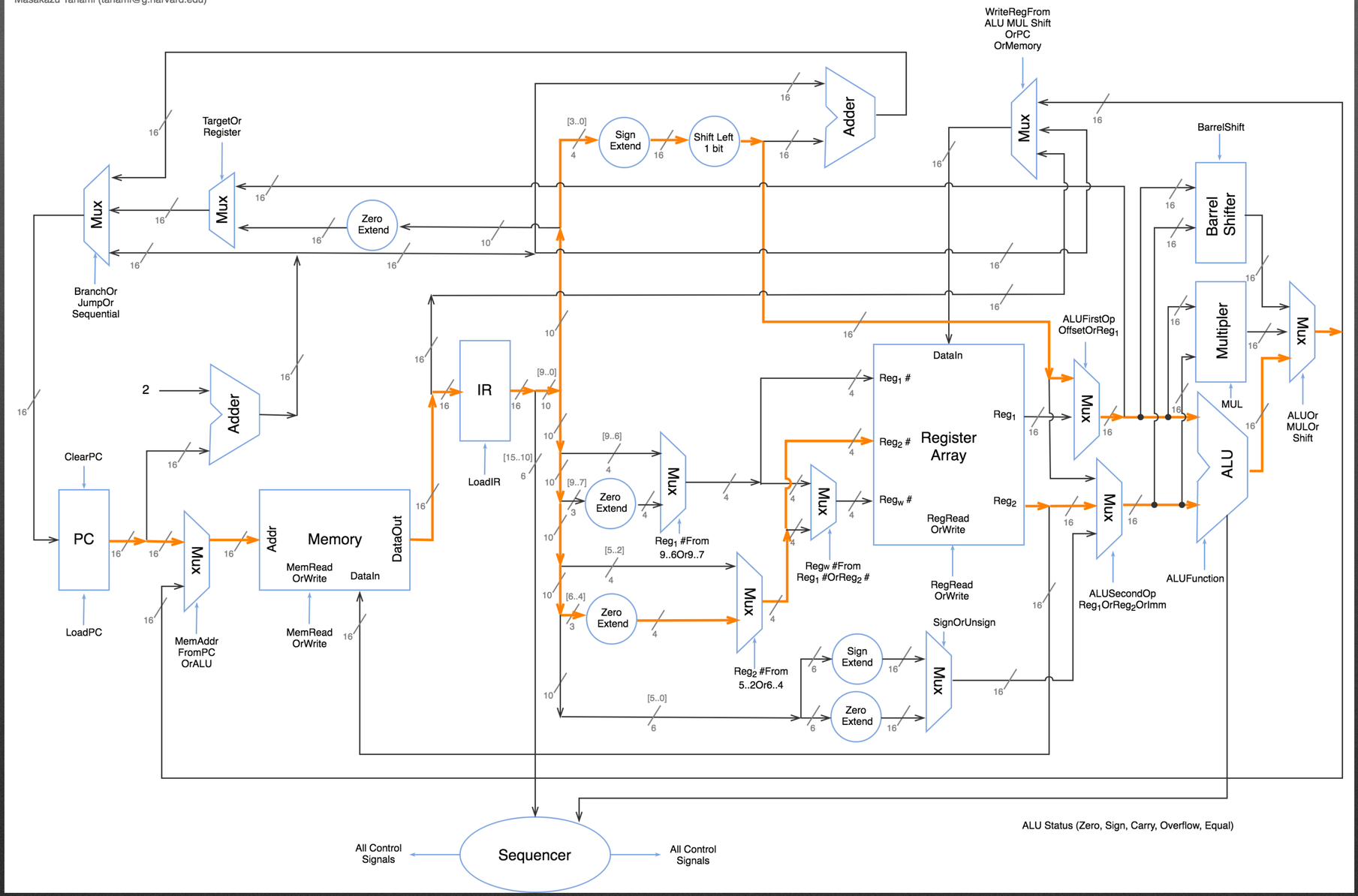
BNE $rd, $rs, 2



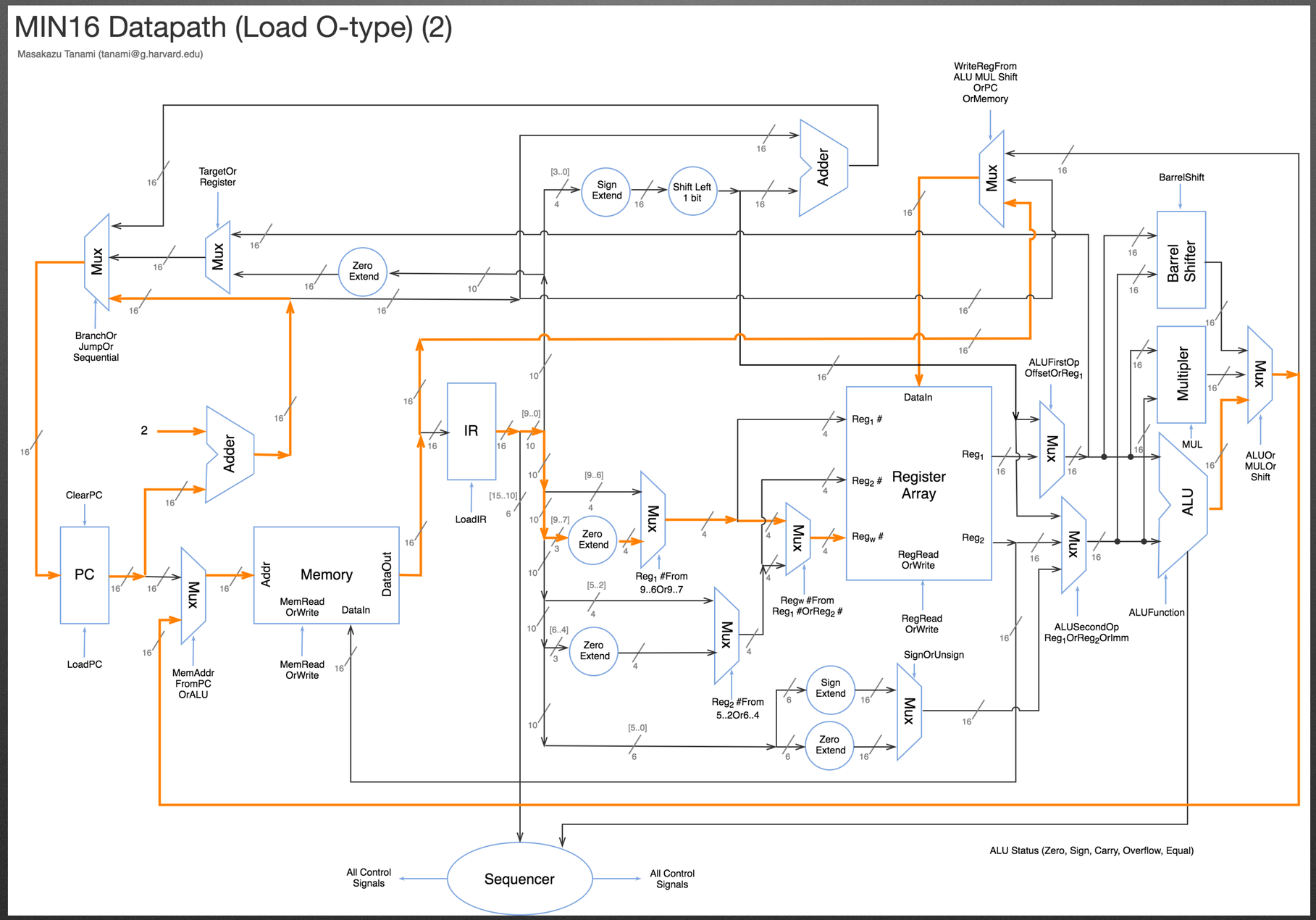
LW $rd, $rs, 2

MIN16 Datapath (Load O-type) (1)

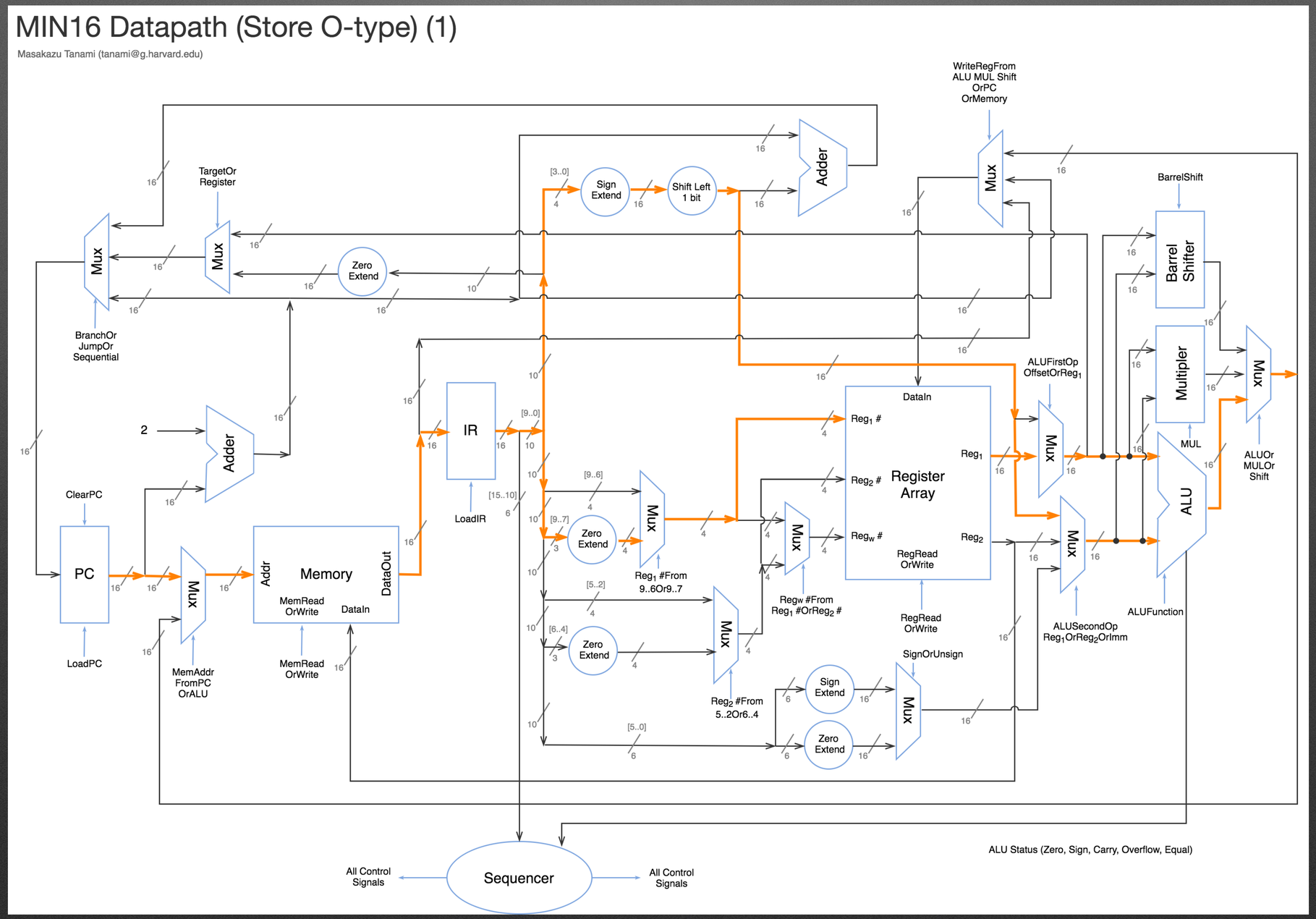
Masakazu Tanami (tanami@g.harvard.edu)



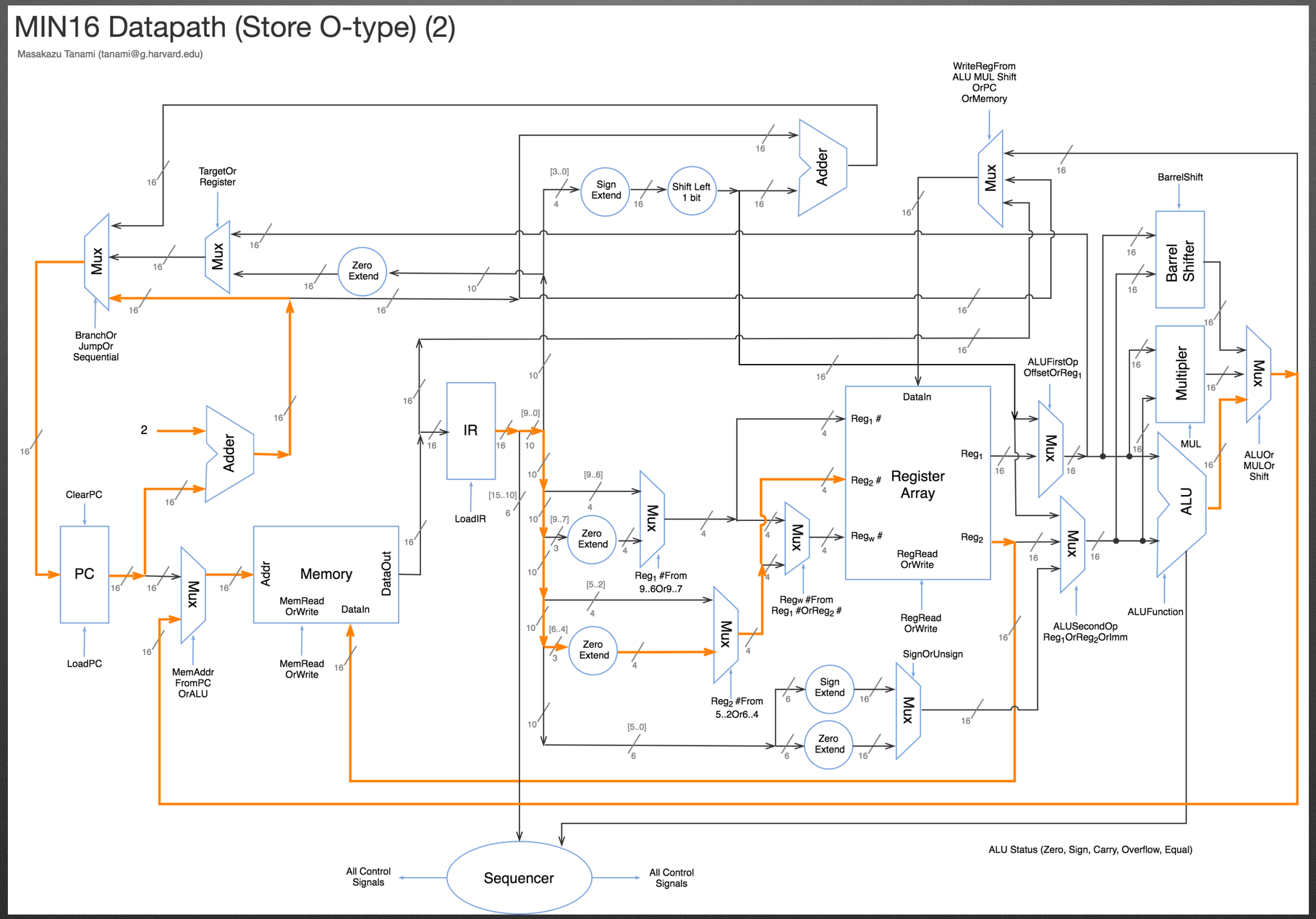
LW $rd, $rs, 2


MIN16 Datapath (Load O-type) (2)
Masakazu Tanami (tanami@g.harvard.edu)
WriteRegFrom ALU MUL Shift OrPC OrMemory
TargetOr Register
BranchOr JumpOr Sequential
Sign Extend
Shift Left 1 bit
Adder
Zero Extend
Mux
BarrelShift
Barrel Shifter
Multipler
MUL
ALU
ALUOr MULOr Shift
ALUFirstOp OffsetOrReg1
ALUSecondOp Reg1OrReg2OrImm
ALUFunction
SignOrUnsign
RegRead OrWrite
DataIn
Register Array
Reg1 #
Reg2 #
Regw #
Reg1
Reg2
Reg1 #From 9..6Or9..7
Regw #From Reg1 #OrReg2 #
Reg2 #From 5..2Or6..4
IR
LoadIR
ClearPC
PC
LoadPC
MemAddr FromPC OrALU
Memory
Addr
DataOut
MemRead OrWrite
ALU Status (Zero, Sign, Carry, Overflow, Equal)
Sequencer
All Control Signals
All Control Signals

SW $rd, $rs, 2


MIN16 Datapath (Store O-type) (1)
Masakazu Tanami (tanami@g.harvard.edu)
WriteRegFrom ALU MUL Shift OrPC OrMemory
TargetOr Register
BranchOr JumpOr Sequential
Zero Extend
Sign Extend
Shift Left 1 bit
Adder
Mux
BarrelShift
Barrel Shifter
Multipler
ALU
MUL
ALUOr MULOr Shift
ALUFirstOp OffsetOrReg1
ALUSecondOp Reg1OrReg2OrImm
ALUFunction
SignOrUnsign
DataIn
Reg1 #
Reg2 #
Regw #
Reg1
Reg2
Register Array
RegRead OrWrite
Reg1 #From 9..6Or9..7
Regw #From Reg1 #OrReg2 #
Reg2 #From 5..2Or6..4
IR
LoadIR
2
ClearPC
PC
LoadPC
MemAddr FromPC OrALU
Addr
Memory
MemRead OrWrite
DataIn
DataOut
Sequencer
All Control Signals
ALU Status (Zero, Sign, Carry, Overflow, Equal)

SW $rd, $rs, 2


MIN16 Datapath (Store O-type) (2)
Masakazu Tanami (tanami@g.harvard.edu)
WriteRegFrom ALU MUL Shift OrPC OrMemory
TargetOr Register
BranchOr JumpOr Sequential
Sign Extend
Shift Left 1 bit
Adder
Zero Extend
Mux
BarrelShift
Barrel Shifter
Multipler
MUL
ALU
ALUOr MULOr Shift
ALUFirstOp OffsetOrReg1
ALUSecondOp Reg1OrReg2OrImm
ALUFunction
SignOrUnsign
DataIn
Reg1 #
Reg2 #
Regw #
Register Array
Reg1
Reg2
RegRead OrWrite
Reg1 #From 9..6Or9..7
Regw #From Reg1 #OrReg2 #
Reg2 #From 5..2Or6..4
IR
LoadIR
ClearPC
PC
LoadPC
2
Adder
MemAddr FromPC OrALU
Addr
Memory
DataOut
MemRead OrWrite
DataIn
ALU Status (Zero, Sign, Carry, Overflow, Equal)
Sequencer
All Control Signals
All Control Signals

The Instruction Set

```
00: R-Type (Arithmetic, Logical, Shift)
0000 00 ADD   0x0 0       ADD $rd, $rs
0000 01 SUB   0x0 1       SUB $rd, $rs
0000 10 MUL   0x0 2       MUL $rd, $rs
0000 11 SLT   0x0 3       SLT $rd, $rs

0001 00 ADDU  0x1 0       ADDU $rd, $rs
0001 01 SUBU  0x1 1       SUBU $rd, $rs
0001 10 MULU  0x1 2       MULU $rd, $rs
0001 11 SLTU  0x1 3       SLTU $rd, $rs

0010 00 AND   0x2 0       AND $rd, $rs
0010 01 OR    0x2 1       OR  $rd, $rs
0010 10 XOR   0x2 2       XOR $rd, $rs
0010 11 NOR   0x2 3       NOR $rd, $rs

0011 00 SLL   0x3 0       SLL $rd, $rs
0011 01 SRL   0x3 1       SRL $rd, $rs
0011 10 SLA   0x3 2       SLA $rd, $rs
0011 11 ROTL  0x3 3       ROT $rd, $rs
```

```
01: I-Type (Immediate)
0100 00 ADDI  0x4 0       ADDI $rd, imm
0100 01 SUBI  0x4 1       SUBI $rd, imm
0100 10 MULI  0x4 2       MULI $rd, imm
0100 11 SLTI  0x4 3       SLTI $rd, imm

0101 00 ADDIU 0x5 0       ADDIU $rd, imm
0101 01 SUBIU 0x5 1       SUBIU $rd, imm
0101 10 MULIU 0x5 2       MULIU $rd, imm
0101 11 SLTIU 0x5 3       SLTIU $rd, imm

0110 00 ANDI  0x6 0       ANDI $rd, imm
0110 01 ORI   0x6 1       ORI  $rd, imm
0110 10 XORI  0x6 2       XORI $rd, imm
0110 11 NORI  0x6 3       NORI $rd, imm

0111 00 SLLI  0x7 0       SLLI $rd, imm
0111 01 SRLI  0x7 1       SRLI $rd, imm
0111 10 SRAI  0x7 2       SRAI $rd, imm
0111 11 ROTLI 0x7 3       ROTI $rd, imm
```

```
10 : ORJ-Type (Jump/Branch Flow Control)
1010 00 J     0xA 0       J    target   (J-type)
1010 01 JAL   0xA 1       JAL  target   (J-type)
1010 10 JR    0xA 2       JR   $rd      (R-type)
1010 11 JALR  0xA 3       JALR $rd, $rs (R-type)

1011 00 BEQ   0xB 0       BEQ  $rd, $rs, offset (O-type)
1011 01 BNE   0xB 1       BNE  $rd, $rs, offset (O-type)
```

```
11: OR-TYPE (Register/Memory Data Move)
1100 00 LW    0xC 0       LW $rd, $rs, offset (O-type)
1100 01 LB    0xC 1       LB $rd, $rs, offset (O-type)
1100 10 SW    0xC 2       SW $rd, $rs, offset (O-type)
1100 11 SB    0xC 3       SB $rd, $rs, offset (O-type)

1101 00 MFHI  0xD 0       MFHI $rd (R-type)
1101 01 MFLO  0xD 1       MFLO $rd (R-type)
1101 10 MTHI  0xD 2       MTHI $rd (R-type)
1101 11 MTLO  0xD 3       MTLO $rd (R-type)
```

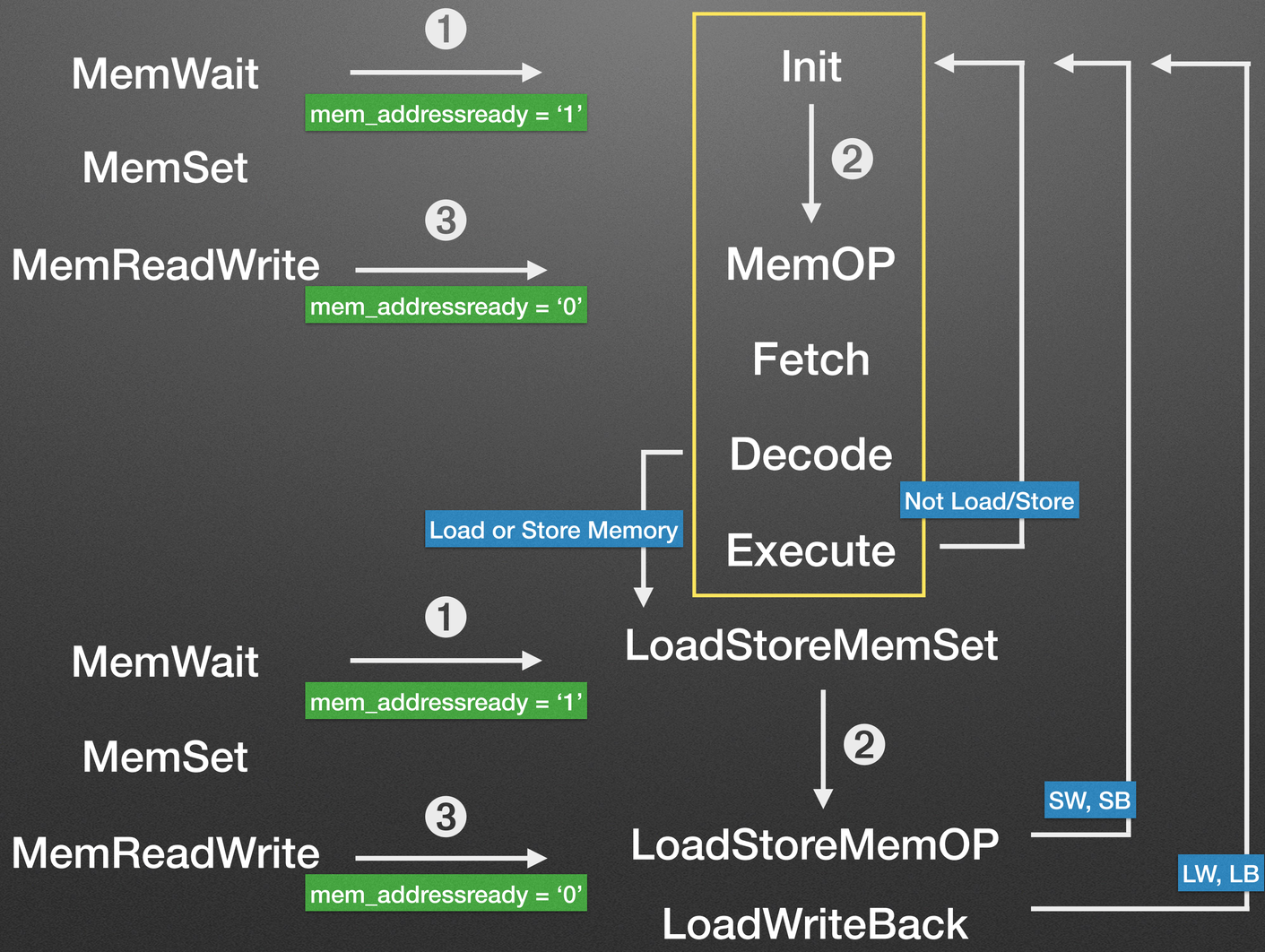
Sequencing Logic

- Control lines are determined by the sequencer (seq.vhd)
- The opcode + function code (opfunc: 6bits) identifies instructions
- ALU and Jump related control lines are determined only by opfunc
- Branch control depends on opfunc and Zero Flag
- Load/Store control lines are chosen by opfunc and the state of FSM

Clocking Scheme

- FSM for Memory and CPU is the core of computation
- Rising edge: State transitions on CPU FSM
- Falling edge: Write Back to registers
- Memory FSM is implemented separately (memio.vhd)

FSM for Memory and CPU



Assembler

- 1st path: build label list
- Repeated address resolution to associate the correct address for labels
- 2nd path: encode line and build error list when found
- Reports: label list and error list
- Human-friendly MIF file (binary, original code with line number)
- Auto-gen to handle 16-bit value for imm(6-bits), offset(4-bits), target(10-bits)
- Address resolution is necessary because each label address depends on all used labels which can cause auto-gen.

```
ADDI $at, tag1 (undefined)
ADDI $at, tag2 (undefined)
tag1:
```

```
DEPTH = 32768;
WIDTH = 16;
ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;
CONTENT
BEGIN
    0000 : 6080;     --   [0b 0110 0000 1000 0000] -> [42: ANDI  $sp, 0]
                     --   auto-gen (0xf000 > 5bits) <- [43: ORI   $sp, 0xf000]
    0001 : 22c0;     --   [0b 0010 0010 1100 0000] -> [asm: (R2):  AND   $t1, $r0]
    0002 : 2040;     --   [0b 0010 0000 0100 0000] -> [asm: (R2):  AND   $at, $r0]
    0003 : 66fc;     --   [0b 0110 0110 1111 1100] -> [asm:  (I):  ORI   $t1, 0x3c]
    0004 : 72ca;     --   [0b 0111 0010 1100 1010] -> [asm:  (I):  SLLI  $t1, 0xa]
    0005 : 246c;     --   [0b 0010 0100 0110 1100] -> [asm: (R2):  OR    $at, $t1]
    0006 : 22c0;     --   [0b 0010 0010 1100 0000] -> [asm: (R2):  AND   $t1, $r0]
    0007 : 66c0;     --   [0b 0110 0110 1100 0000] -> [asm:  (I):  ORI   $t1, 0x0]
    0008 : 72c6;     --   [0b 0111 0010 1100 0110] -> [asm:  (I):  SLLI  $t1, 0x6]
    0009 : 246c;     --   [0b 0010 0100 0110 1100] -> [asm: (R2):  OR    $at, $t1]
    000a : 22c0;     --   [0b 0010 0010 1100 0000] -> [asm: (R2):  AND   $t1, $r0]
    000b : 66c0;     --   [0b 0110 0110 1100 0000] -> [asm:  (I):  ORI   $t1, 0x0]
    000c : 246c;     --   [0b 0010 0100 0110 1100] -> [asm: (R2):  OR    $at, $t1]
    000d : 2484;     --   [0b 0010 0100 1000 0100] -> [asm: (R2):  OR    $sp, $at]

    000e : 60c0;     --   [0b 0110 0000 1100 0000] -> [44: ANDI  $fp, 0]
    000f : 24c8;     --   [0b 0010 0100 1100 1000] -> [45: OR    $fp, $sp]
                     --   label: 0010 <- [48: BIT_SERIAL_INPUTREADY:  .equ 1]
```

```
 70   AND  $ra, $rb, $rc
 71
putchar             : address is 0x[003a] 0x[001d] (mif)
putcharloop         : address is 0x[0112] 0x[0089] (mif)
str_to_int          : address is 0x[0346] 0x[01a3] (mif)
stringToInt         : address is 0x[02f6] 0x[017b] (mif)
zeroHandler         : address is 0x[0242] 0x[0121] (mif)
--
TOTAL LINES : 47
[-- ERROR LIST REPORT --]
Wrong Format: AND  $ra, $rb, $rc    : line 70
```

Emulator

- Three modes to run
  - 0: Simple Mode
    1: Display Register Mode
    2: Line-by-Line Execution Mode
- Debug Information
  - MIF Word address
  - MIF Line (HEX machine code)
  - Memory Address (Program Counter)
  - Addressing Modes
  - Disassembled assembly code
  - Register Content highlighting changed value
  - Status Flag Register

```
Press 'm' for Multiplier, 's' for BarrelShifter: m

= Multiplier =
1st: 3
2nd: 5
Ans: 15
1st: 6
2nd: -9
Ans: -54
```

```
[0055:cf70] (00aa)  (O):  SB    $rc, $rd, 0x0
[stdout] word [0041] at address [ff04] is char [A]

 r0   at   sp   fp   ra   rb   rc   rd   s0   s1   t0   t1   hi   lo   pc   fl
0000 0002 eff4 f000 0002 ff00 ff04 0041 e018 051a 0041 0004 0000 0000 00aa 0000

[0056:c0a0] (00ac)  (O):  LW    $at, $sp, 0x0
 r0   at   sp   fp   ra   rb   rc   rd   s0   s1   t0   t1   hi   lo   pc   fl
0000 051a eff4 f000 0002 ff00 ff04 0041 e018 051a 0041 0004 0000 0000 00ac 0000

[0057:6240] (00ae)  (I):  ANDI          $s1, 0x0
 r0   at   sp   fp   ra   rb   rc   rd   s0   s1   t0   t1   hi   lo   pc   fl
0000 051a eff4 f000 0002 ff00 ff04 0041 e018 0000 0041 0004 0000 0000 00ae 0001  ZF

[0058:2644] (00b0) (R2):  OR    $s1, $at
 r0   at   sp   fp   ra   rb   rc   rd   s0   s1   t0   t1   hi   lo   pc   fl
0000 051a eff4 f000 0002 ff00 ff04 0041 e018 051a 0041 0004 0000 0000 00b0 0000

[0059:c0a1] (00b2)  (O):  LW    $at, $sp, 0x1
 r0   at   sp   fp   ra   rb   rc   rd   s0   s1   t0   t1   hi   lo   pc   fl
0000 e018 eff4 f000 0002 ff00 ff04 0041 e018 051a 0041 0004 0000 0000 00b2 0000

[005a:6200] (00b4)  (I):  ANDI          $s0, 0x0
 r0   at   sp   fp   ra   rb   rc   rd   s0   s1   t0   t1   hi   lo   pc   fl
0000 e018 eff4 f000 0002 ff00 ff04 0041 0000 051a 0041 0004 0000 0000 00b4 0001  ZF

[005b:2604] (00b6) (R2):  OR    $s0, $at
 r0   at   sp   fp   ra   rb   rc   rd   s0   s1   t0   t1   hi   lo   pc   fl
0000 e018 eff4 f000 0002 ff00 ff04 0041 e018 051a 0041 0004 0000 0000 00b6 0002      SF
```

Barrel Shifter

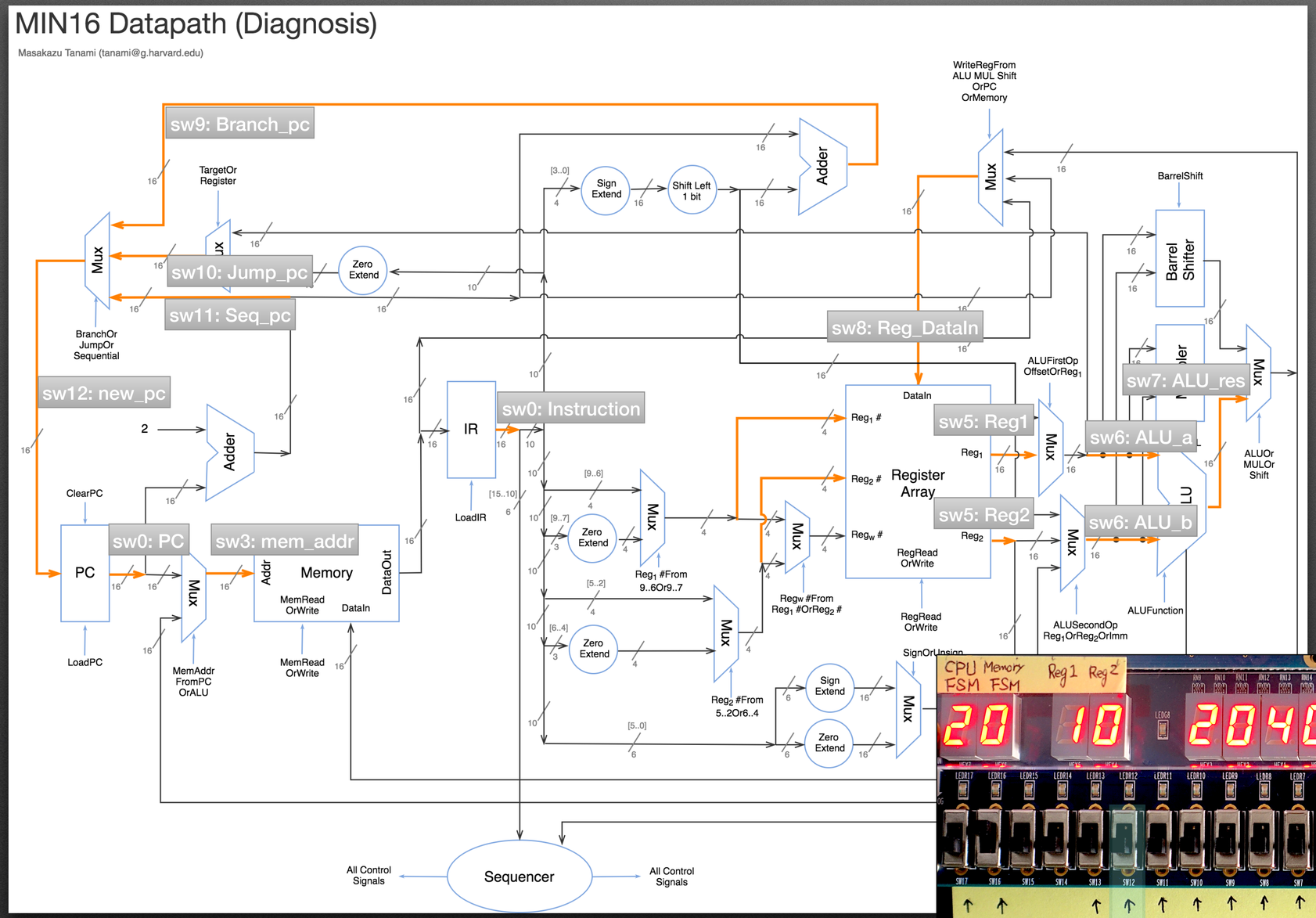
- Current Implementation is done by VHDL loop and bit-by-bit copy.
- Intend to implement using hardware merge sort algorithm described in the design doc (MIN16_Instruction_Set.pdf).
- Cost Analysis using merge sort
  - The number of bit-by-bit sort is n/2 times
    The number of division is log(n)
    The cost per merge sort is n/2*log(n)
    Barrel Shifter using two merge sorts costs n*log(n)
  - Bit-by-bit sort is done by multiplexer
    Barrel Shifter requires n*log(n) numbers of multiplexer
    MIN16 computer needs 64 multiplexers to compose the Barrel Shifter
    Combined multiplexers achieve n-bit shift within a single clock cycle

```
-- ROTL "1111"
rotateLeft: process(rd, rs)
  variable index_tmp:  integer;
  variable index_end: integer;
begin
  res_ROTL <= zero_vec;
  index_tmp := 15 - index_end;
  index_end := to_integer(unsigned(rs));
  for index in 15 downto 0 loop
    res_ROTL(index) <= rd(index_tmp);
    index_tmp := index_tmp - 1;
    if index = index_end then
      index_tmp := 15;
    end if;
  end loop;
end process rotateLeft;
```

Next Step

- Hardware Barrel Shifter
- Hardware Multiplier
- Carry-Lookahead Full Adder
- SHA-256 Hash Algorithm Implementation

CSCI E-93 Computer Architecture

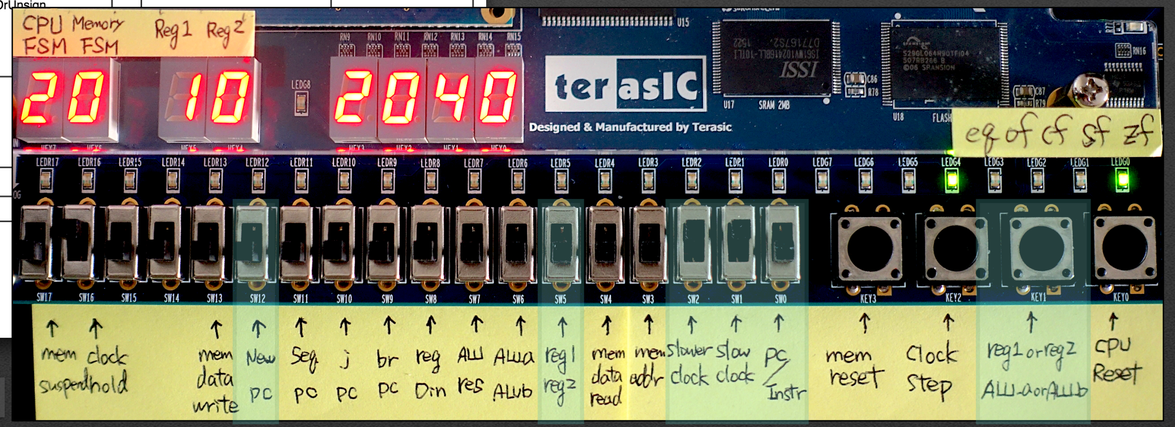



```
Diagnostic Pin Assignments

key0: cpu reset
key1: reg1 or reg2, ALU_a or ALU_b
key2: clock_step
key3: mem_reset

sw0:  pc/instr
sw1:  slow clock
sw2:  slower clock
sw3:  result <= mem_addr
sw4:  result <= mem_data_read
sw5:  result <= reg1 or reg2 value
sw6:  result <= ALU_a or ALU_b
sw7:  result <= ALU_res
sw8:  result <= reg_din
sw9:  result <= branch_pc
sw10: result <= jump_pc
sw11: result <= seq_pc
sw12: result <= new_pc
sw13: result <= mem_data_write

sw16: clock_hold
sw17: mem_suspend
```




```
2040: 0b 0010 0000 0100 0000 -> AND $at, $r0
```

Demo

```
Environment:
Quartus II 32-bit Version on Linux Ubuntu 32-bit
VirtualBox running on MacOSX
Serial port emulator for Mac: Serial (decisivetactics.com)
```

Thank you!

Masakazu Tanami
tanami@g.harvard.edu