CS147 - Lecture 11

Kaushik Patra
(kaushik.patra@sjsu.edu)

1

- Putting together an ALU
- Putting together a Processor

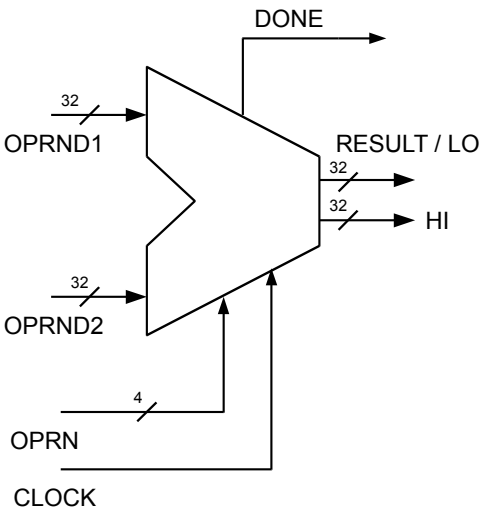
Reference Books:

1) *Chapter 4 of 'Computer Organization & Design by Hennessy, Patterson*

Putting together an ALU ...

2

Sequential ALU Interface



- Supports mathematical and logical function between two operands.
- 'DONE' is the flag that indicates the requested operation is done.

OPRN	Operation	Function
0x0	NoOp	No Operation
0x1	Addition	RESULT = OPRND1 + OPRND2
0x2	Subtraction	RESULT = OPRND1 - OPRND2
0x3	Multiplication	{HI, LO} = OPRND1 * OPRND2
0x4	Division	{HI, LO} = OPRND1 / OPRND2; HI = Remainder, LO = Quotient
0x5	Bitwise AND	RESULT = OPRND1 & OPRND2
0x6	Bitwise OR	RESULT = OPRND1 | OPRND2
0x7	Bitwise NOR	RESULT = ~(OPRND1 | OPRND2)
0x8	Set Less Than	RESULT = OPRND1 < OPRND2
0x9	Shift Left Logical	RESULT = OPRND1 << OPRND2
0xA	Shift Right Logical	RESULT = OPRND1 >> OPRND2

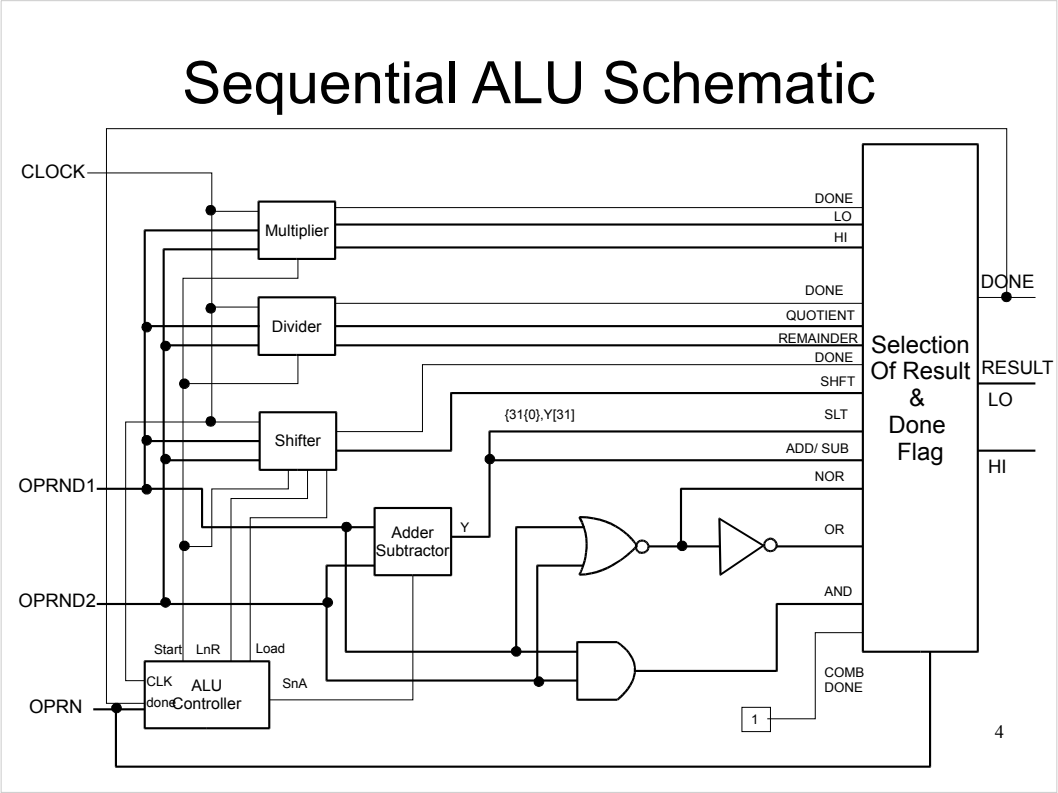
3

- A sequential ALU implements shift, multiplication and division operation with sequential digital circuit. As a result the ALU is not guaranteed to produce a result within one clock cycle. As we have seen in sequential version of these functions (mul, div and shift) it will take variable amount of clock cycle to complete the operation. For example multiply takes 31 iteration, division takes 32 iteration and shift takes number of shift needed. On the other hand all the combinational functions (add, sub, and, or, nor) completes in one clock cycle. In this variable completion time scenario, to synchronize the operation with the other components on a processor, sequential ALU must have a 'DONE' flag to indicate that the operation is done. Once this flag is set to one, the components, waiting on the ALU result can consume the result.

- Being sequential ALU, it needs a clock signal to synchronize its operation.

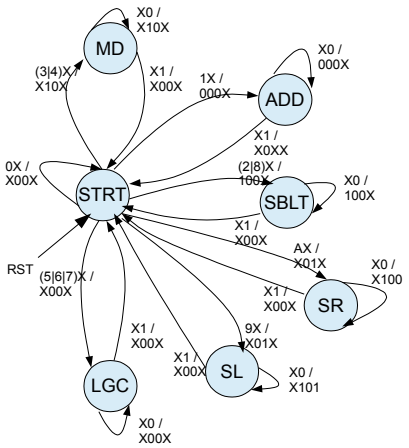
- It has two output to accommodate results for multiplication and division.

- There is a NoOp ALU operation (no operation) which means no operation from ALU. Addition of this operation makes it easier to identify if there is any operation requested.



- This is a very basic schematic for an sequential ALU. This does not contain very minute details of the logic circuit. However, this is good enough to convey the basic idea.
- There are individual components for each operations. All these components produces some result. The output result is selected from all these results depending on the operation requested. The DONE flag is also selected depending on the operation requested.
- The sequential circuit needs some 'START' control signal to start the operation. Once the 'START' is 1, multiplication, division and sequential shifter starts its operation. Once done, they raise the DONE flag. However, if the START is down in middle, they abort the operation.
- Additionally shifter has 'load' signal and 'LnR' signal to load data and dictate left of right shift during the operation.
- To control various control signal values, a small controller is in place which takes the operation code (OPRN) and DONE signal out of ALU.
- Since the combination part completes in one clock cycle, they all uses a common static '1' signal as done signal.
- The adder/subtractor unit also needs another control signal SnA to select between addition and subtraction operation.
- It is also possible to share the add/sub unit with the multiplier and divider unit to optimize resource on the ALU circuit.

ALU Controller State Diagram



- Input = {OPRN, done} in hex
- Output = {SnA, start, load, LnR} in binary
- Modes
  - STRT : Start state
  - MD : Multiplication / Division state
  - ADD : Add state
  - SBLT : Subtract / SLT state
  - SR : Shift Right state
  - SL : Shift Left state
  - LGC : Logic state

5

- The ALU controller is a state machine which starts from 'STRT' state and stays there till the OPRN code remains at NoOp (0). Upon the operation requested, depending on the operation code it goes into MD (for mul, div), ADD (for add), SUB (for sub, slt), SR (for shift right), SL (for shift left), LGC (logic operations). The controller stays in one of this state till the operation is done. Once done, it gets back to STRT state. Each of the state transition generates appropriate control signal.

Combinational ALU

- Except for the division operation, all other functions can be implemented using combination circuit design methodology.
  - This will not need any sequential control unit.
  - This will not need to produce any DONE signal.
  - One clock cycle operation – may be slower overall.
  - Possibly need small decoder to create SnA and LnR signal.

6

- The minimum mathematical operation an ALU needs to support is the addition and subtraction. The multiplication and division can be implemented in software, if needed, using just additional and subtraction.
- The components which are depending on the ALU result can use the result in one clock cycle since it is guaranteed to have the result within one clock cycle.
- Since the operations are accommodate within one clock cycle, the clock period must be equal or more than the time needed to complete the slowest operation.

Putting together a processor ...

7

Data & Control Path

- Processor Signals
  - Data signal to carry information to be processed or already processed.
  - Control signal to control the data flow through the circuit.
- Data path consists of the components which perform arithmetic operations and stores data and signal connection between data.
- Control path consists of the components which generates and route control signal to various parts of data path to control the flow of data.

8

- To understand processor operation and architecture, it is very important to understand the concept of data path and control. There are two types of signal flows through a processor circuit. The data signal carries the input data, output data and intermediately processed data. Data flows through the processor, transforming on its way to final output. The control signal controls the flow of data through the processing elements so that the requested operation is done.

Designing Data Path

- Common components
  - Program Counter Register (PC)
  - Stack Pointer Register (SP)
  - Instruction Register (INST)
  - Register File
  - ALU
  - Memory (for both instruction and data)
- Identify data path for each instruction and operation using common components.
- Combine data path using multiplexer if data path in conflict.

9

- To construct a simple processor, only a handful of components are used. There are register files to store intermediate results, special registers (PC, SP, INST, etc.), ALU and Memory.
- The very first step to design a processor is to review all the instructions that the processor is going to implement. Processor design is closely dependent on the targeted set of instruction. Each unique instruction set will need a unique design to support it.
- For each instruction the data flow path identified. In this process we may club together operations which are similar syntactically and semantically.
- Once the data paths are identified, all of them are merged to use common elements. Any conflict (data source or destination) is to be resolved using multiplexer to select appropriate data flow for the operation.

Designing Controller

- Identify control signals including common components and multiplexers used to combine data path for different instructions and operations.
- Controller is a state machine cycles through the following state.
  - Instruction fetch (IF)
  - Instruction Decode / Register Fetch (ID/RF)
  - Instruction Execute (EX)
  - Memory Access (MEM)
  - Write Back (WB)
  - Additional NOP / WAIT state is needed if a step is not completed in one clock cycle.
- Each state will create appropriate control signal depending on what instruction is executed.

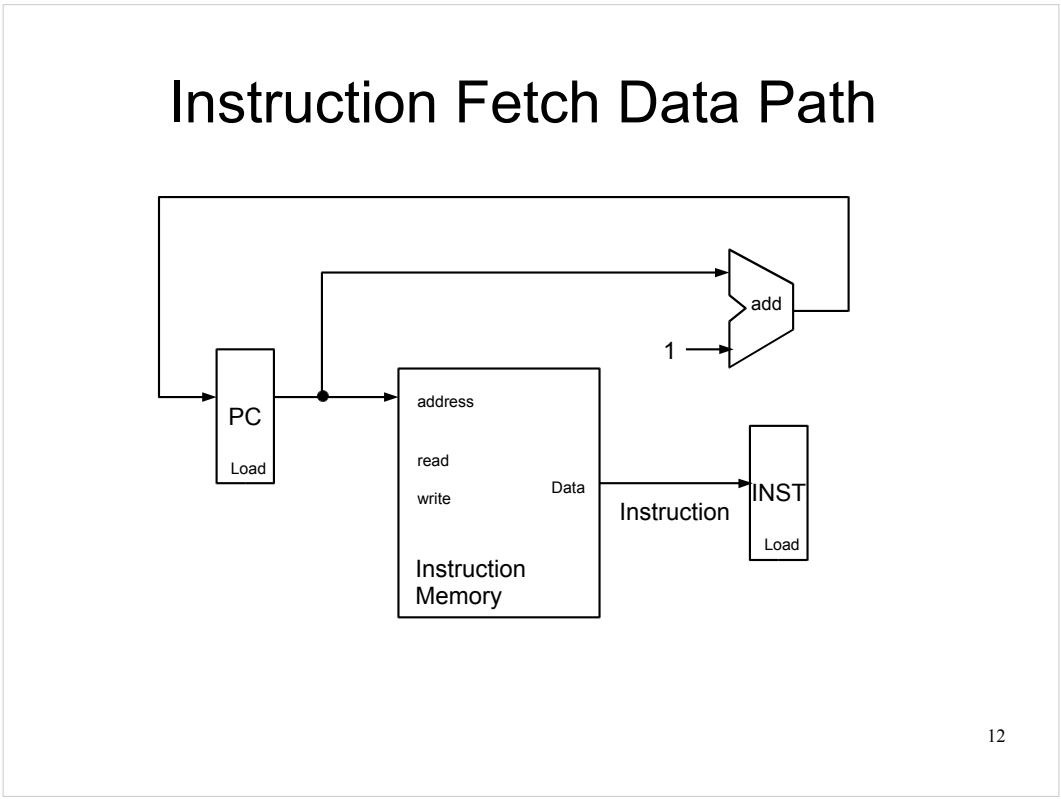
10

- The control signals includes the following.
  - ALU operation signal.
  - Register file's read/write signal.
  - Memory's read/write signals.
  - Special registers' load signals.
  - Selection signal for various mux needed combine data path.

- For MIPS like design, the controller continuously cycles through 5 stages as listed. The following are the functions at each stage.
  - IF is for setting the memory address for next instruction and to issue read signal to memory.
  - ID is for registering the fetched instruction and decode the instruction and issue the control signals for corresponding operation. At this point fetching of register content is also done.
  - EX is for the execution of the command. At this stage, mostly the ALU operation is done (including data memory address calculation).
  - MEM is the stage to access the memory (if needed). For MIPS style instruction set, only LW or SW instruction needs this stage. However, for uniformity of the design, we assume all instruction goes through this stage (and actually may not to anything).
  - WB is the stage to write the result back from the ALU or memory to register file.

Instruction Fetch Data & Control ...

11



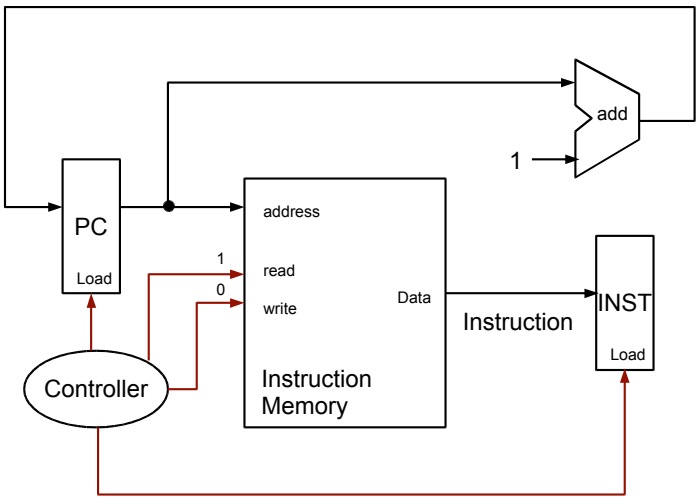
- The instruction fetch data path involves the following components.
  - Program counter (PC)
  - Memory (instruction storage part)
  - Instruction register (INST)
  - Adder to increment the PC content to point to the next instruction.

- PC is initialized with power up address at start. Instruction is read in from this addressed and store in instruction register at ID phase.

- Since we are assuming word addressable memory, where two adjacent address will access two adjacent words (32-bit in size), each time the PC needs to be incremented by 1. The combinational adder does this job to increase the value of PC by 1.

- In a byte addressable memory where two adjacent memory address will access two words overlapping into 3 byte positions (LSB or MSB depending on endianness of the system – big endian or little endian).

Instruction Fetch Control Signals

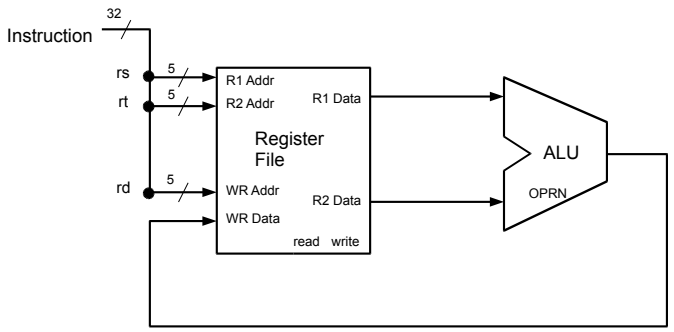


13

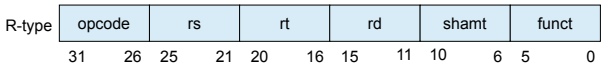
- In this system there are three control signals for the IF stage – memory read signals (mem_read=1 and mem_write=0), PC load signal. The memory is accessed with the current address of PC.
- At the ID stage, the INST register is loaded with the read data from the instruction section of the memory.
- At the WB phase the next address is loaded into the PC.

R-type Instruction Data & Control ...

General R-type Instruction Data Path



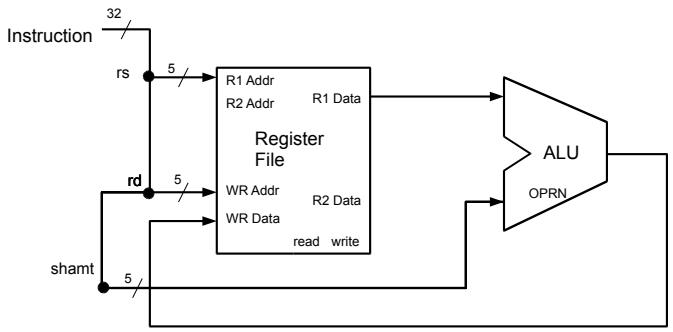
Operation: R[rd] = R[rs] (op) R[rt]



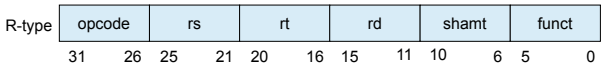
15

- The general R-type data path involves the following components.
  - Register file
  - ALU
- As per the R-type instruction coding is done, there is no decoding needed for the operands and the result address in the register file. The filed rs, rt and rd are are directly taken and connected to the R1, R2 read address and write address respectively.
- The two read results R1-data and R2-data are connected to input of the ALU.
- The output of the ALU is connected back into the write data port of the register file.

Shift Instruction Data Path



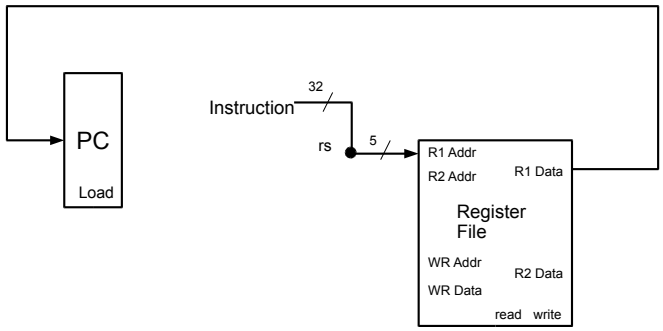
Operation: R[rd] = R[rs] (op) shamt



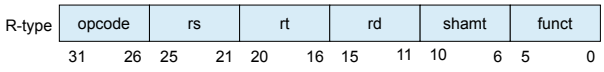
16

- Data path for the shift operation is little bit different to the other R-type operation. The second operand comes directly from the 'shamt' field instead of the R2-read address data value.

Jump Reg Instruction Data Path



Operation: PC = R[rs]

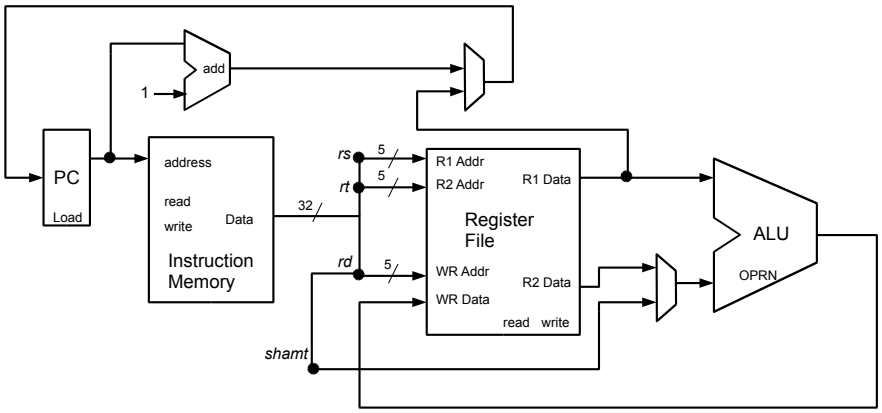


17

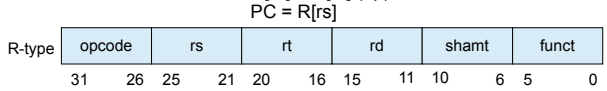
- Jump to register content (jr) instruction data path is different than other R-type instructions. This instruction makes the PC set to the value containing in the target register 'rs'.

- In this operation the register content is read and loaded into the PC, so that the next instruction fetched will be from the location as dictated by the content of the rs register.

Combined R-type Instruction Data Path



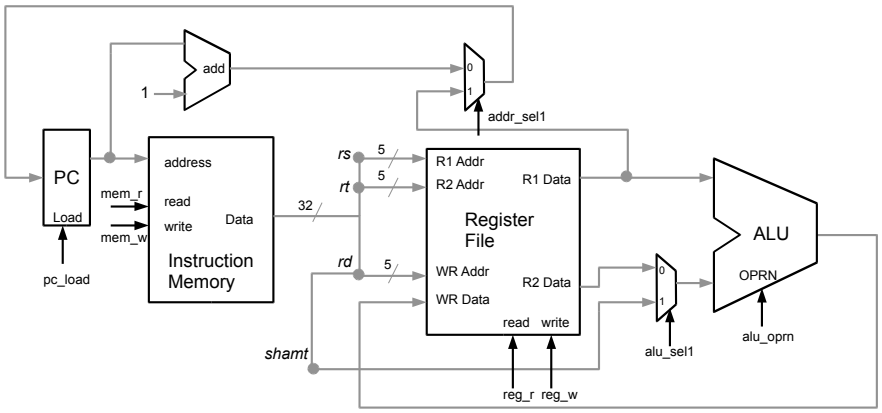
Operation: R[rd] = R[rs] (op) R[rt]
R[rd] = R[rs] (op) shamt
PC = R[rs]



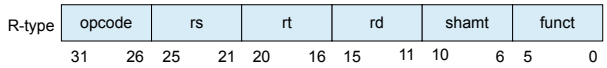
18

- In combining general R-type instruction, shift instruction and the J-type instruction along with the instruction fetch data path we can observe the following conflicts.
  - Next address source for PC – either from increment or from the 'rs' register value.
  - The second operand of the ALU can either come from 'rt' register value or from the 'shamt' value filed in the instruction.
- We uses mux at the conflicted data source point to select data source from alternate sources.

Combined R-type Instruction Control



Operation: R[rd] = R[rs] (op) R[rt]
R[rd] = R[rs] (op) shamt
PC = R[rs]



19

- The following are the control signals for the R-type instructions.
  - pc_load to load new data into PC
  - addr_sel1 to select between incremented PC address or 'rs' register content.
  - alu_sel1 to select between 'rt' register content or 'shamt' from the instruction as the second operand of the ALU.
  - OPRN signal for the ALU to select the ALU operation.
  - Register read and write signals reg_r, reg_w.

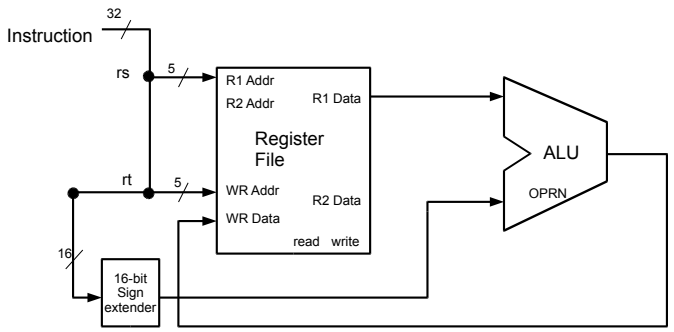
- At the ID/RF phase, the OPRN is set corresponding to the instruction. The register file's reg_r and reg_w signal is configured to read from the register file.

- At EX phase alu_sel1 is set to select the data from the required source for the corresponding instruction.

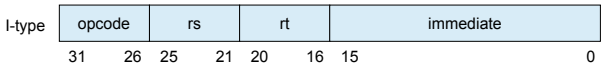
- At the WB phase the register file's reg_r and reg_w signal is configured to write ALU result into the target register if the instruction is not 'jr'. For the 'jr' instruction the addr_sel1 is set to select the next address from the register file output.

I-type Instruction Data & Control ...

Arithmetic I-type Instruction Data Path



Operation: R[rt] = R[rs] (op) SignExtImm



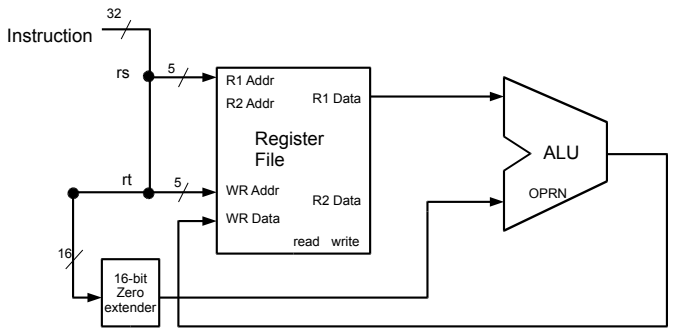
21

- I-type instructions differs from R-type instructions in terms of the source of the second operands. For I-type instruction the 2nd operands comes directly from the immediate filed of the instruction code. The destination register address comes from the 'rt' field, unlike the R-type operations where the destination register address comes from the 'rd' field.

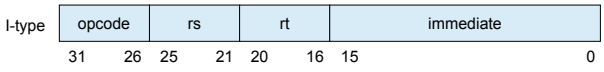
- For arithmetic I-type operation this immediate 16-bit field is extended with the sign bit to make it a 32-bit number. The sign extension is easily done with repeating the last bit for another 16 bit lines by connecting them to the last bit of the 16-bit number.

- The data path components are same as the R-type with addition of the 16-bit sign extender to add 16 sign bits to the original number.

Logical I-type Instruction Data Path



Operation: R[rt] = R[rs] (op) ZeroExtImm

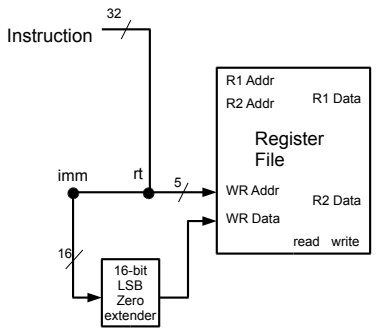


22

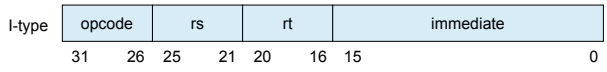
- For logical I-type operation this immediate 16-bit field is extended with another 16 bits of 0. This is 16-bit zero extender. For logical operation we do not need sign extension.

- The data path components are same as the R-type with addition of the 16-bit zero extender to add 16 bits of 0 to the original number.

LUI Instruction Data Path



Operation: R[rt] = {imm, 16b'0}

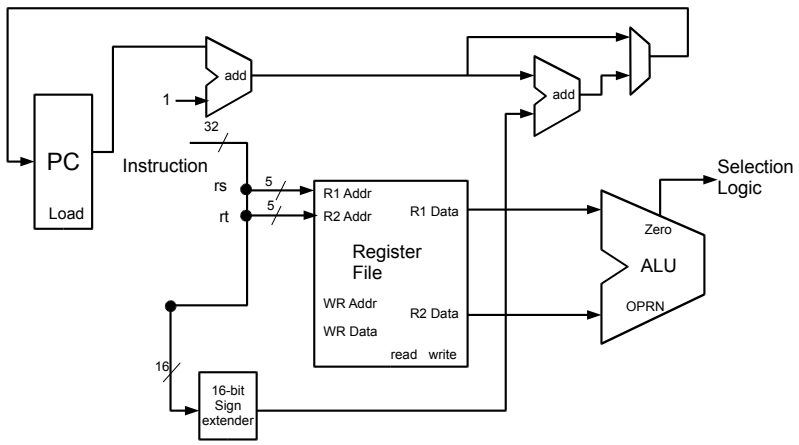


23

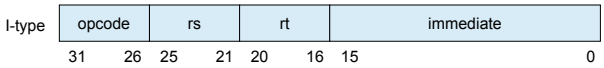
- For LUI (Load Upper Immediate) the 16-bit immediate is transformed into a 32-bit number with upper 16-bit containing the immediate filed and lower 16-bit containing all 0 bits. This transformed number is then stored into the destination register as defined in 'rt'.

- The data path components are only the register file and the 16-bit LSB zero extender.

Branch I-type Instruction Data Path



Operation: PC = PC + 1 + SignExtImm;
if R[rs] == R[rt] or R[rs] != R[rt]



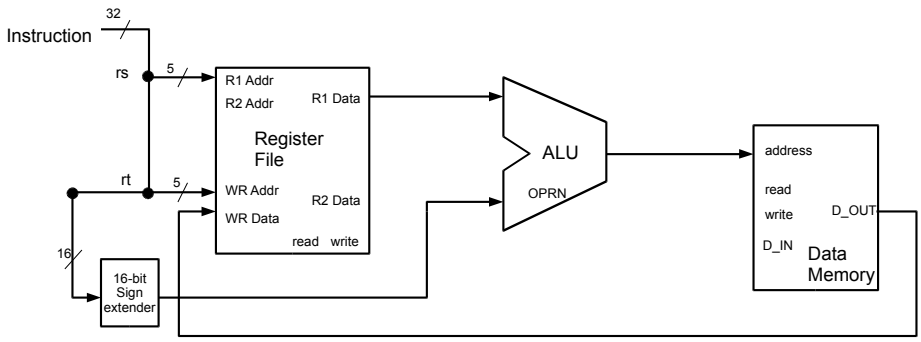
24

- For I-type branch operations branch address is computed by adding PC+1 with the sign extended immediate filed (this enables jump forward and backward from one decision point). The selection of next address between (PC+1) or (PC+1+SignExtImm) is done with a mux with selection control signal generated for Branch-On-Equal and Branch-On-NoEqual. (PC+1+SignExtImm) is done with an extra adder.

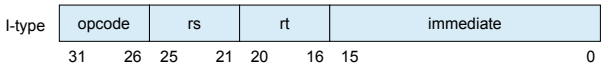
- The content of 'rs' and 'rt' register are subtracted and zero flag is tested. If the zero flag is on, the content of 'rs' and 'rt' are equal, otherwise not. ALU unit usually provides the zero flag too. If it does not, it is easy to implement by doing a NOR between all the bits of the result. Only if all the bits are zero, then the zero flag will be turned on.

- If ZeroFlag is true and operation is 'beq' then (PC+1+SignExtImm) is selected. On the other hand, if ZeroFlag is false and operation is 'bne' then (PC+1+SignExtImm) is selected. (PC+1) is selected in all other cases.

LW Instruction Data Path



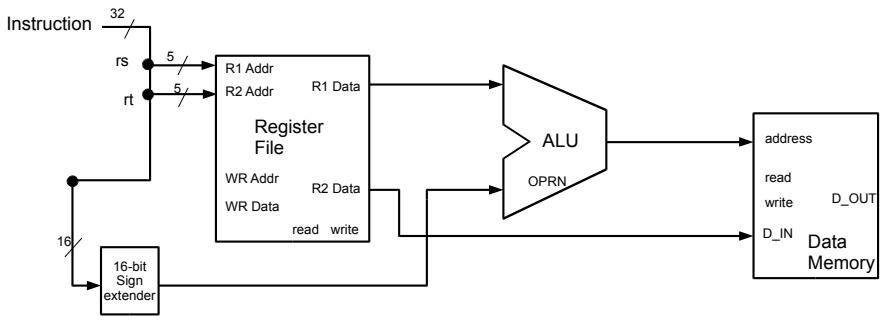
Operation: R[rt] = M[R[rs] + SignExtImm]



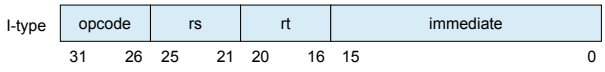
25

- For load word operation, the data address is computed by adding the 'rs' register content and the sign extended immediate. The read data is stored back into the register 'rt'.

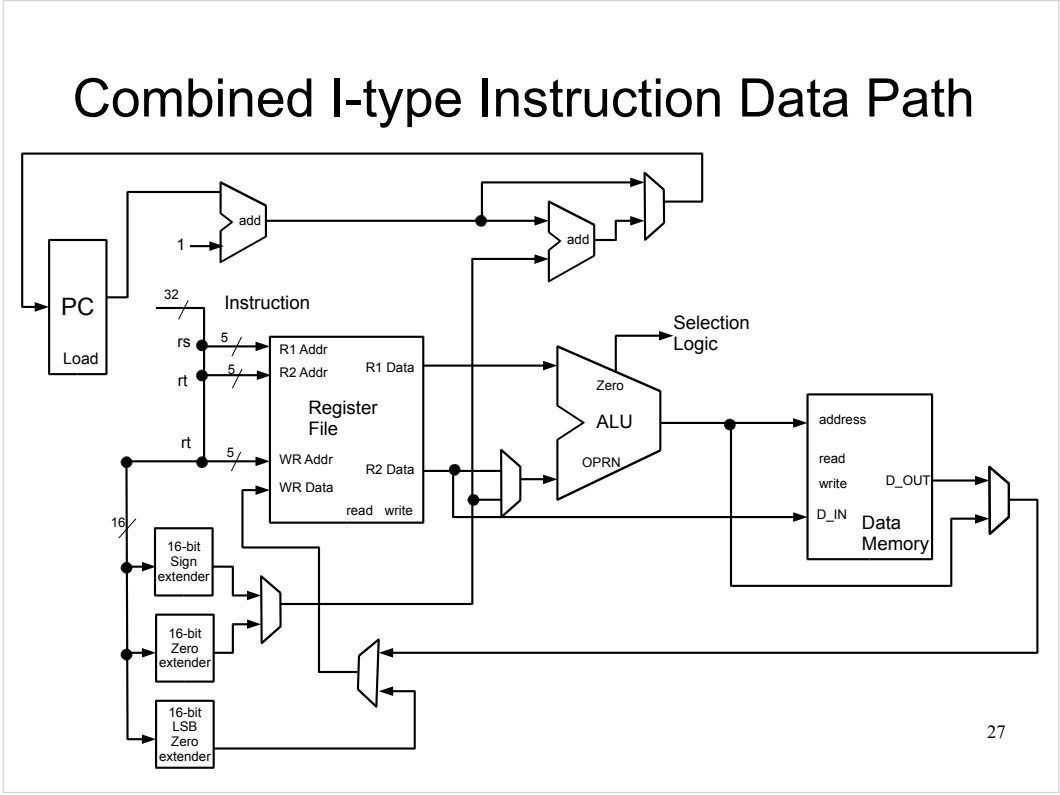
SW Instruction Data Path



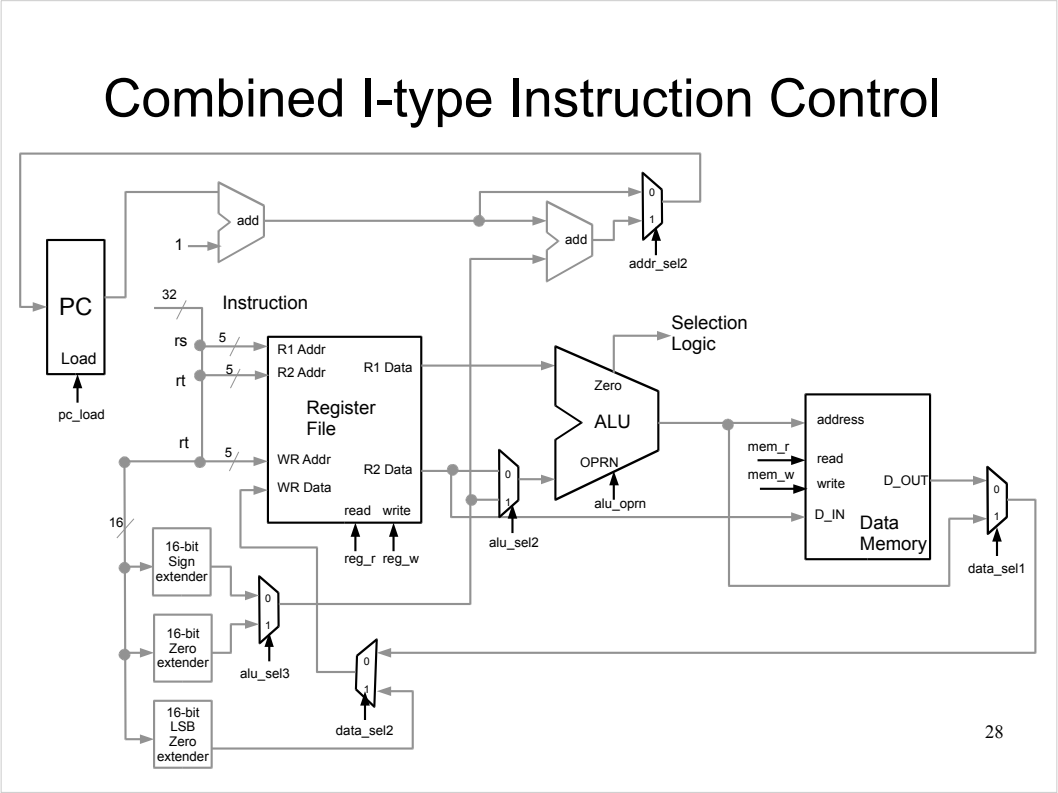
Operation: M[R[rs] + SignExtImm] = R[rt]



- For store word operation, the data address is computed by adding the 'rs' register content and the sign extended immediate. The write data is the content of register 'rt'.



- While combining I-type instruction data path the following conflicting data paths are identified.
    - Data source to ALU and Branch address adder between sign extended and zero extended number.
    - Data source of 2nd operand to ALU as the register content or the immediate filed (sign extended or not).
    - The next address choosing between (PC+1) and (PC+1+SignExtImm).
    - The write data source to the register file can be either ALU result, or memory read data or the 16-bit LSB zero extender (for LUI operation).
    -
- We uses mux at the conflicted data source point to select data source from alternate sources.



- The following are the control signals for the R-type instructions.
  - pc_load to load new data into PC
  - addr_sel2 to select between incremented PC address or computed branch address.
  - alu_sel2 to select between 'rt' register content or 'immediate' from the instruction (sign extended or not) as the second operand of the ALU.
  - alu_sel3 to select between sign extension and zero extension of the 'immediate' field of the instruction.
  - data_sel1 to select between ALU and memory read data.
  - data_sel2 to select LSB zero extended immediate if needed.
  - OPRN signal for the ALU to select the ALU operation.
  - Register read and write signals reg_r, reg_w.
  - Memory read and wirte signals mem_r, mem_w.

- At the ID/RF phase appropriate control signal setting is applied to reg_r, reg_w.
-
- At EXE phase OPRN, alu_sel2, alu_sel3.

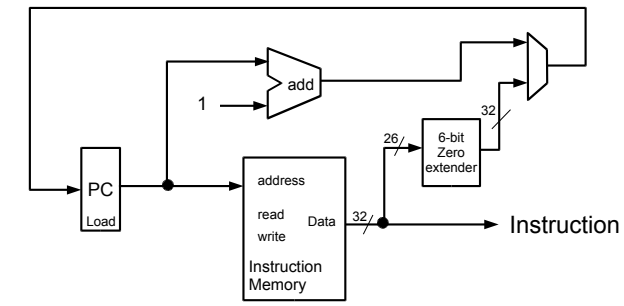
- At the MEM phase, appropriate signal is set for mem_r and mem_w if needed.

- At the WB phase appropriate signal value is set for reg_r, reg_w, add_sel2, pc_load, data_sel1 and data_sel2.

J-type Instruction Data & Control ...

29

JMP Instruction Data Path

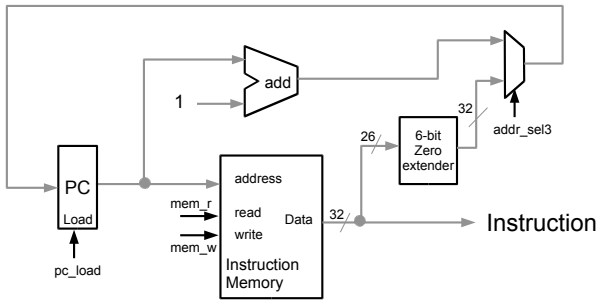


Operation: PC = {6'b0, address}

30

- Jump to given address in the address filed (26-bit) needs to be extended to 32-bit with all 0 in the MSB positions. The address can directly be derived from the instruction code padded with 0 at the 6 MSB positions. There should be multiplexer to select between this address and (PC+1) depending on whether jump instruction is requested or not.

JMP Instruction Control

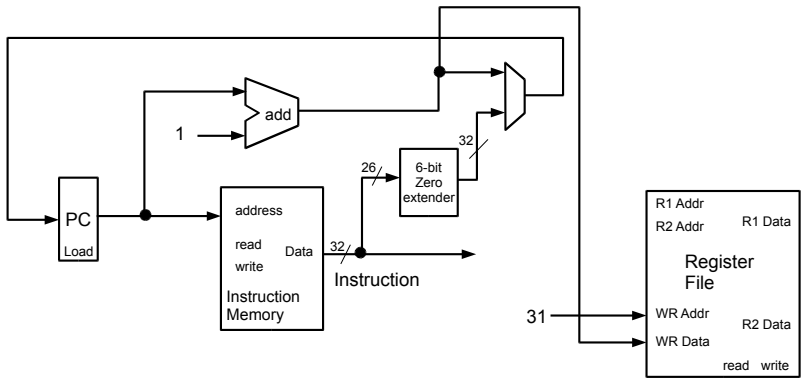


Operation: PC = {6'b0, address}

31

- The major control signals are pc_load and addr_sel3 which is are set appropriately at the WB stage.

JAL Instruction Data Path

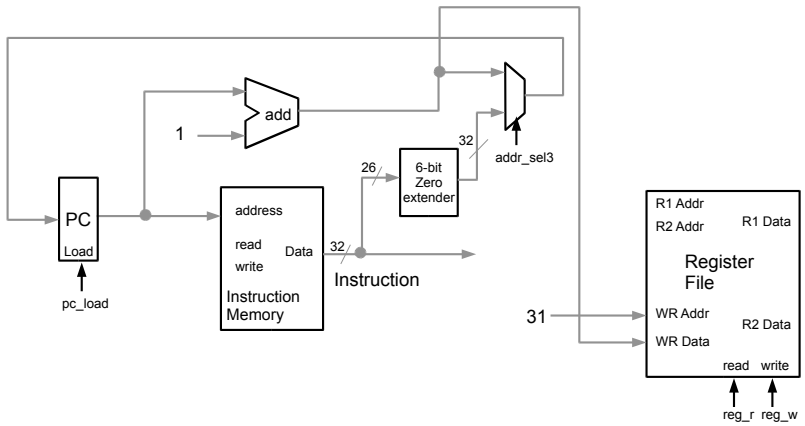


Operation: R[31] = PC + 1; PC = {6'b0, address}

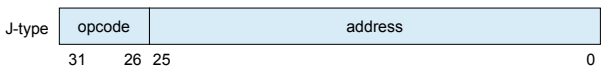
32

- Jump and link instruction data path is an extension of the jump instruction data path. The (PC+1) data is written to register 31 (fixed address) at the WB satge.

JAL Instruction Control



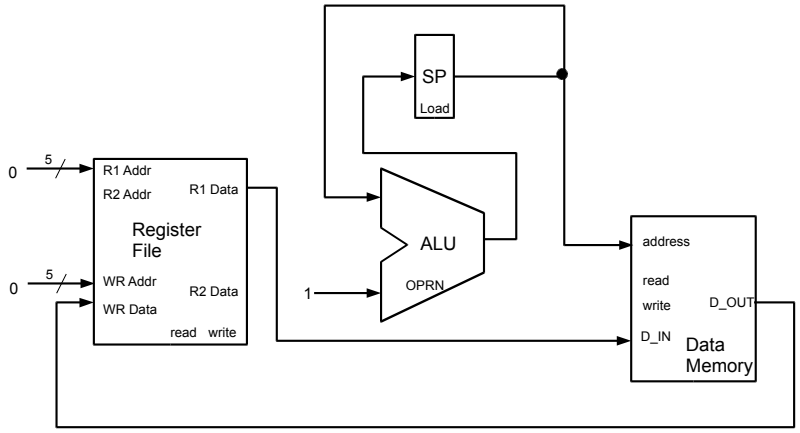
Operation: R[31] = PC + 1; PC = {6'b0, address}



33

- In addition to the jump instruction control reg_r and reg_w is set appropriately at WB stage to configure register file into write mode.

Stack Instruction Data Path

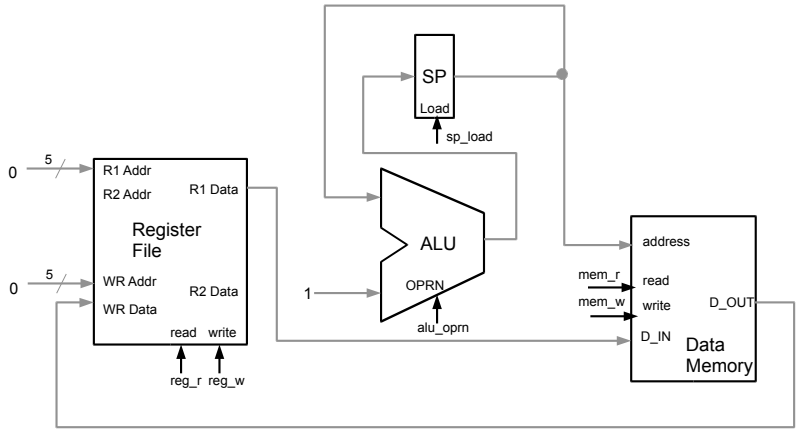


Operation: push – M[$sp] = R[0]; $sp = $sp – 1
pop – $sp = $sp + 1; R[0] = M[$sp]

34

- The stack operation takes memory location of the stack pointer from a special register called stack pointer (SP). The SP content is automatically changed with push and pop operation to maintain the integrity of the stack pointer. Output of the stack register is connected to the address of the memory. The next address of SP comes from ALU (increment or decrement the current SP value). The memory read data is written into register 0 (for pop operation). Memory write data comes from register 0 (for push operation).

Stack Instruction Control



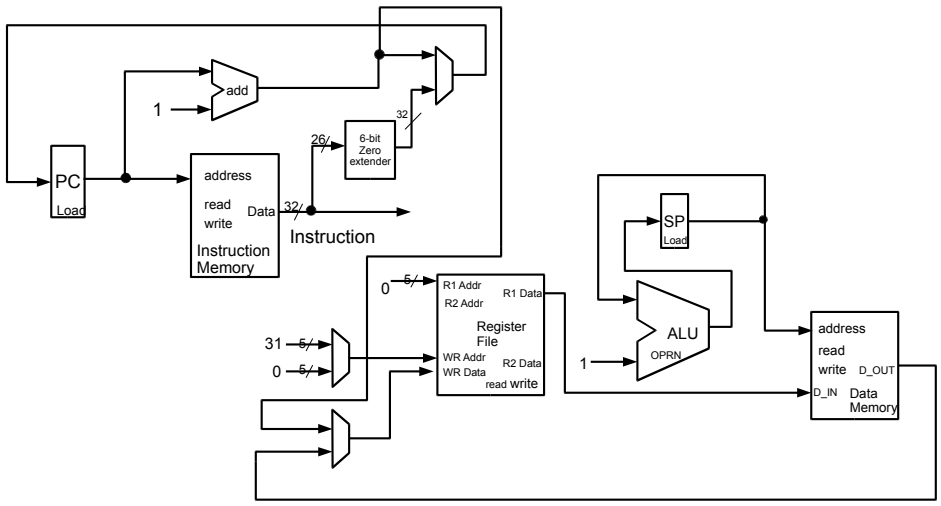
Operation: push – M[$sp] = R[0]; $sp = $sp – 1
pop – $sp = $sp + 1; R[0] = M[$sp]

35

- For push operation, at ID/RF phase the reg_r and reg_w is set to read mode and at EX phase OPRN is set to subtract. At the MEM phase, mem_r and mem_w is set to write mode. At WB phase sp_load is turned on to load the new content to SP register.

- For pop operation, at EX phase, OPRN is set to add and sp_load is set to 1 so that the incremented SP value is loaded first into SP. At MEM phase, mem_r and mem_w is set to read mode. At WB phase the reg_r and reg_w is set to write mode to write the memory content into register 0.

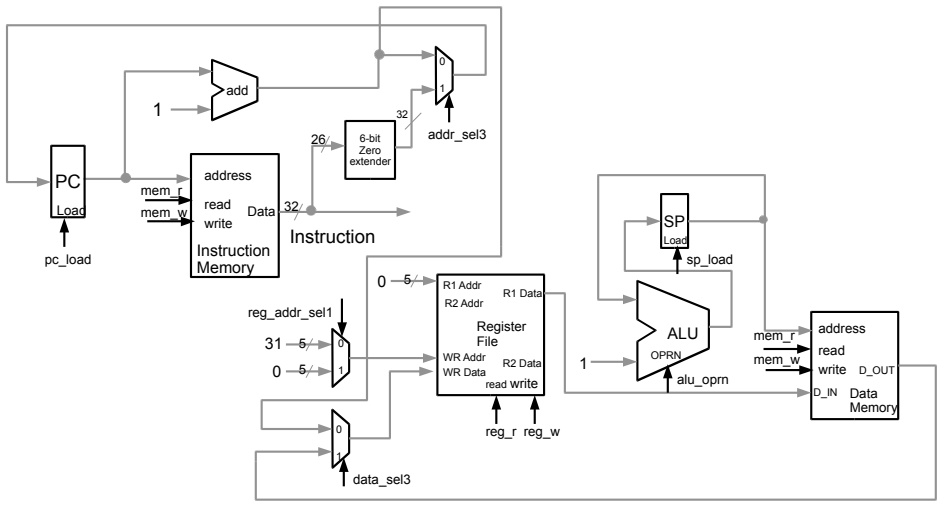
Combined J-Type Instruction Data Path



36

- If we combine the data path of J-type operation we can find three point of data path conflict.
  - Destination address between 0 and 31 (for pop and jal)
  - Register write data source between (PC+1) and memory read data (for jal and pop).
  - Selection between jump address and PC+1 as the next instruction address.
- Like other instructions, J-type instruction data path conflict is resolved by using mux at the conflict point.

Combined J-Type Instruction Control



37

- The following are the control signals.
  - pc_load to load the PC registers.
  - addr_sel3 for selecting next instruction address between (PC+1) and jump address.
  - reg_add_sel1 to select register write address between 0 and 31.
  - data_sel3 to select between PC+1 or memory data out (for jal or pop).
  - reg_r and reg_w to control register file read and write mode.
  - mem_r and mem_w to control between memory read and write mode.
  - sp_load to load the SP register.

- At the ID/RF phase the reg_r and reg_w is set to read mode to retrieve the register data (for push operation)

- At EX phase the next stack address is computed by setting OPRN to appropriate value. The new address is updated to SP if it is pop operation.

- At the MEM phase the stack location in memory is accessed for data read (pop) or data write (push).

- At the WB phase the data is written into register file and the PC and SP (for push operation). The address and data to register file is selected by reg_addr_sel1 and data_sel3. The reg_r and reg_w is also set properly.

Conclusion ...

38

Design Notes

- Processor can be implemented as single clock cycle system as long as it does not support divide operation. Each step (IF, ID, EXE, MEM, WB) will need one clock cycle to complete.
- However, single clock cycle system tends to be slower because one clock cycle needs to cover longest delay path in the system.
- System architecture is greatly influenced by the target instruction set.

39

- By building the processor what we have observed is that the components used and connection between them are greatly influenced by the target instruction set. At each step the actual instruction from cs147sec05 is consulted to build up the final data path. For every unique instruction set, these combinations will be unique to that set of instrcution.

CS147 - Lecture 11

Kaushik Patra
(kaushik.patra@sjsu.edu)

- Putting together an ALU
- Putting together a Processor

Reference Books:

1) *Chapter 4 of 'Computer Organization & Design by Hennessy, Patterson*

