Hardware Design I Chap. 10 Design of microprocessor

Computing Architecture Lab.
Hajime Shimada
E-mail: shimada@is.naist.jp

Outline

- What is microprocessor?
- Microprocessor from sequential machine viewpoint
  - Microprocessor and Neumann computer
  - Memory hierarchy
  - Instruction set architecture
- Microarchitecture of the microprocessor
  - Microarchitecture with sequential processing
  - Microarchitecture with pipelined processing

What is microprocessor?

- LSI for processing data at the center of the computer
  - Also, called “processor”
- There are several type of microprocessors
  - Central Processing Unit (CPU)
  - Microcontroller
  - Graphic accelerator
  - Other several accelerators

Central Processing Unit (CPU)

- A nucleus of Neumann computer
  - Detail will be taught in later slide
- Sometimes, the word “microprocessor” denotes this
- By combining CPU with memories, disks, I/Os, we can create PC or server
- Examples: Intel core i7, Fujitsu SPARC64 VII, AMD Opteron, ...

Microcontroller

- A processor used for control of electric devices
  - Optimized for those use
    - e.g. give high current drive ability to output pin to directly drive LED
- Many of them can organize computer with one chip
  - Implement memory hierarchy into them
- Example: Renesus H8, Atmel AT91, Zilog Z80, ...
  - Too many companies provide them


Graphic accelerator

- A processor used for processing graphic
  - Also called Graphic Processing Unit (GPU)
- Implement too many ALU to utilize parallelism
  - In graphic processing, usually, we can process each pixel independently
  - It also utilized for high parallelism arithmetic
- Examples: NVIDIA GeForce, AMD(ATI) Radeon, ...

Other accelerators

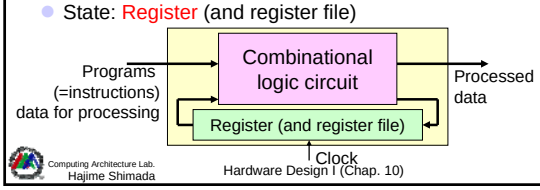
- There's several processor to accelerate data processing which is not suitable to process with CPU or GPU
  - But recently, GPU intrudes to this area
- Usually, it implements much ALU to supply high arithmetic performance
- Example: ClearSpeed CSX, Ageia PhysiX, ...

Outline

- What is microprocessor?
- Microprocessor from sequential machine viewpoint
  - Microprocessor and Neumann computer
  - Memory hierarchy
  - Instruction set architecture
- Microarchitecture of the microprocessor
  - Microarchitecture with sequential processing
  - Microarchitecture with pipelined processing

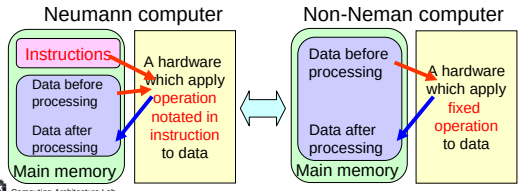
Microprocessor from sequential circuit viewpoint

- We can abstract microprocessor with following sequential machine
- Inputs
  - Programs (= instructions)
  - Data for processing
- Outputs: Processed data
- State: Register (and register file)



Neumann computer

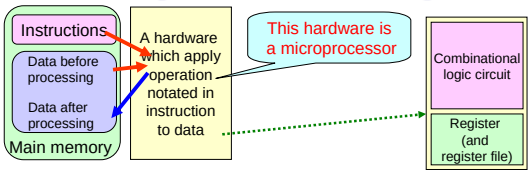
- Neumann computer is current major organization
- Point of Neumann computer
  - Instructions and data are placed in main memory
  - Instructions manipulate data of flip-flop and memories
  - We can apply different manipulation by different instruction



Advantages and disadvantages of Neumann computer

- Advantages
  - We don't have to modify hardware between different data processing
    - EDSAC(1949) is one of the early Neumann computer
    - ENIAC have to change wire connection if it change processing
  - We can execute complicated processing with multiple instructions
- Disadvantages (Neumann bottleneck)
  - Communication between processor and memory increases
  - Slow memory drags down processor performance
- How about non-Neumann computer?
  - It remains in some specific use (e.g. movie codec)
  - It begins to reposition with reconfigurable hardware ->Chap. 9

Neumann computer and microprocessor



- Microprocessor is a hardware which operate data processing in Neman computer
  - Also called "processor" or "Central Processing Unit (CPU)"
  - It includes a part of main memory (see memory hierarchy)
- Hardware organization of the microprocessor differs between its purpose
  - For server, for PC, for high performance embedded, for embedded, ...

States in the microprocessor

- How we define states of sequential machine in the processor?
  - Usually, we call it register
- There are many types of registers
  - Special purpose registers (SPR)
    - Program counter (PC): Denotes position of instruction which is executing
    - Flag register: Denotes carry generation, overflow, ...
  - Global purpose registers (GPR)
    - Used for hold data before/after processing (work as a part of main memory)
    - Also, used for intermediate data under arithmetic
- The organization of register differs between instruction set architectures

Relationship between GPR and memory hierarchy is shown in later

Inputs for the microprocessor

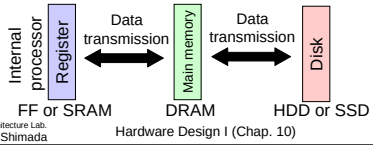
- There are two inputs of sequential machine in the processor
  - Instruction: must be defined if we design sequential machine
  - Data: don't have to define them
- What's instruction?
  - Series of bits: e.g. 00000000100001010100000000010000
  - Usually, we use assembly language to represent it
    - A programming language which has one to one relationship to instruction
    - It defines operation relationship between registers and main memory (in basic)
    - e.g. add R8, R4, R5 (GPR #8 = GPR #4 + GPR #5)

<-> 00000000100001010100000000010000

Introduce how to define it efficiency in later slide

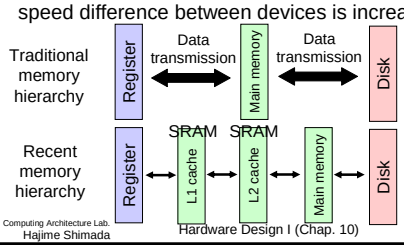
GPR and memory hierarchy (1/2)

- In recent processors, the GPR becomes a part of main memory
  - Firstly the processor moves data from main memory to register
  - Processor apply operation to the data in the register
  - After operation, it write back data to main memory
- This organization effectively reduces workload for main memory
  - Assuming that we apply multiple operation to data



GPR and memory hierarchy (2/2)

- We call "memory hierarchy" for those hierarchical
  - Including disk
  - It also reduces performance degradation from slow device
- Recently, number of hierarchy increasing because the speed difference between devices is increasing



Instruction set architecture (ISA)

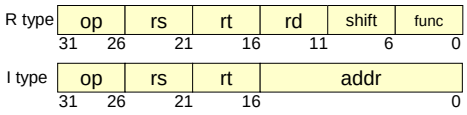
- To create sequential machine, we have to define format of inputs and internal state
  - Internal state: denoted by registers (for internal state)
  - Inputs: instructions
- We usually call this definition as Instruction Set Architecture (ISA)
  - Including systematic instruction construction method
- By defining ISA carefully you can reduce
  - States (registers)
  - Combinational logics

Instruction encoding

- Instruction is encoded to chunk of binary under ISA definition
  - e.g. add R8, R4, R5 (GPR #8 = GPR #4 + GPR #5)

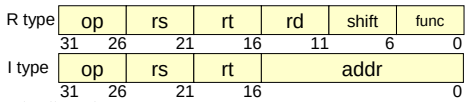
<-> 00000000100001010100000000010000

- In usual encoding, we give meaning into some chunk of bits



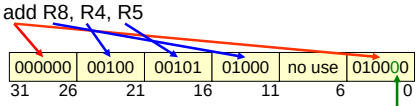
Example of instruction encoding (1/3)

- Example: Instruction encoding of MIPS
- Total length is 32-bit
- It has meaning in several chunk of bits
  - op: Type of operation (arithmetic, load, store, branch, ...)
  - rs: Source operand 1 for arithmetic
  - rt: Source operand 2 for arithmetic
  - rd: Destination operand for arithmetic (store arithmetic result)
  - shift: Amount of shift
  - func: Type of arithmetic (supplemental for op)
  - addr: Immediate value for arithmetic

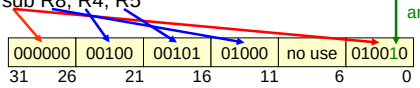


Example of instruction encoding (2/3)

- add R8, R4, R5



  - Operation: R8 = R4 + R5 (add: addition)
- sub R8, R4, R5

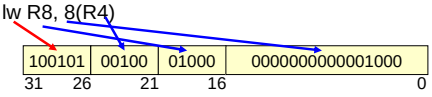


  - Operation: R8 = R4 - R5 (sub: subtract)

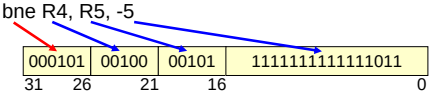
This difference indicates different arithmetic

Example of instruction encoding (3/3)

- lw R8, 8(R4)



  - Operation: Load value in (R4 + 8) position on main memory to R8 (lw: load word)
- bne R4, R5, -5



  - Operation: if R4 != R5, back to 5 prior instruction (bne: branch not equal)

Short Exercise

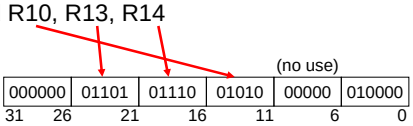
- Let's translate following assembly to instruction notated by binary
  - Refer R-type instruction notation in slides

add R10, R13, R14

Answer

- Let's translate following assembly to instruction notated by binary
  - Refer R-type instruction notation in slides

add R10, R13, R14



Outline

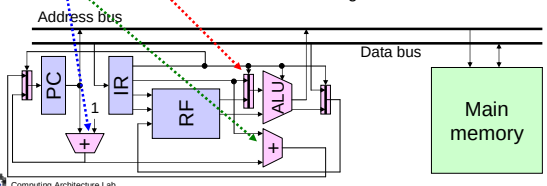
- What is microprocessor?
- Microprocessor from sequential machine viewpoint
  - Microprocessor and Neumann computer
  - Memory hierarchy
  - Instruction set architecture
- Microarchitecture of the microprocessor
  - Microarchitecture with sequential processing
  - Microarchitecture with pipelined processing

What's Microarchitecture?

- An implementation of processor on the hardware
- We can choose several possible microarchitecture in same ISA
  - e.g. Intel Core i7, Intel Atom, AMD Phenon
  - It can execute same program (e.g. Windows) because ISA is the same
- Usually, we choose microarchitecture for the purpose of the computer
  - e.g. Choose low power consumption microarchitecture for notebook PC

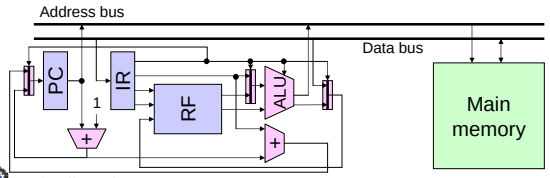
One organization of microprocessor (1/3)

- Combinational logics
  - ALU: execute add, sub, logical arithmetic, shift, ...
  - Multiplexers: construct data path from instructions and values in register
  - Adder after PC: increment PC to indicate next instruction
  - Adder beside ALU: calculate branch target in branch instruction



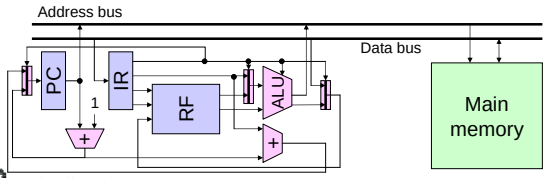
One organization of microprocessor (2/3)

- Buses for main memory
  - Address bus: send address value which indicate read/write position in main memory
  - Data Bus
    - Send data value which is read from main memory
    - Send data value which will be written into main memory



One organization of microprocessor (3/3)

- Registers
  - Register file (RF): chunk of GPR
    - Number is differ between architectures (e.g. 32 in MIPS)
  - Program counter (PC)
  - Instruction register (IR): hold instruction comes from main memory



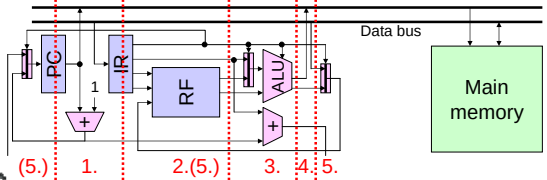
How to understand operation of prior chunk of hardware?

- It seems that it's hard to understand operation of prior large hardware -> True
- How can we understand it easily?

  -> Decompose hardware to 5 part and understand those operation
- This 5 part decomposition has importance in operation
  - Operate 5 part sequentially: 5 phase operation processor
    - Finish one instruction with 5 clock pulse
  - Operate 5 part simultaneously: 5 stage pipelined processor
    - Finish one instruction with 1 clock pulse (in general case)

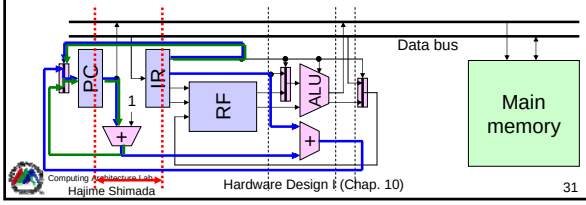
Decomposition to 5 part

1. Instruction fetch (IF)
2. Instruction decode (ID), register read
3. Execution (EX)
4. Memory access (MA)
5. Write back to register (WB), and commit



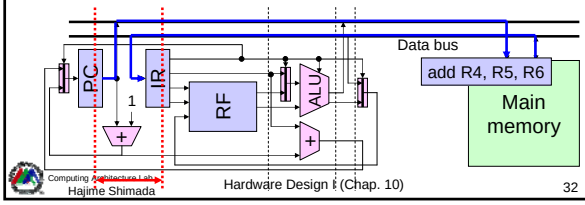
Operation of IF (1/2)

- Manages PC updating
  - Update with incremented PC value
    - If instruction is not branch instructions (or not taken branch)
  - Update with branch target address
    - If instruction is (take conditional) branch instruction
- Read instruction which is indicated by PC value



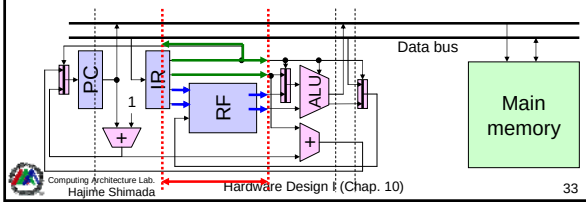
Operation of IF (2/2)

- Manages PC updating
- Read instruction which is indicated by PC value
  - Send content of PC to address bus
  - Capture instruction (to IR) comes from data bus



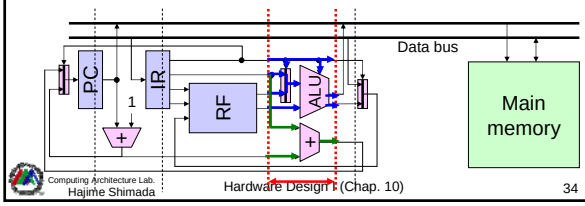
Operation of ID

- Read registers by rs or rt bits in instruction
  - Usually, RF is consist of RAM so that rs or rt becomes address for RAM
- Decode instructions
  - Generate several control signals from instruction
    - e.g. signal for multiplexer before PC



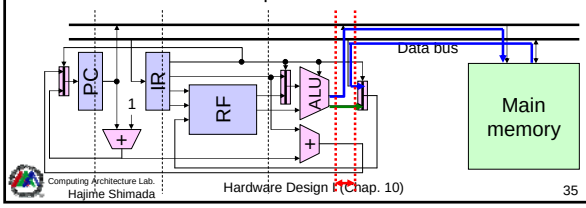
Operation of EX

- Arithmetic
  - Memory address generation on memory access is also operated
  - Detailed arithmetic is indicated by “func” part of instruction
- Calculate branch target address
  - Add PC+1 and immediate value comes from instruction



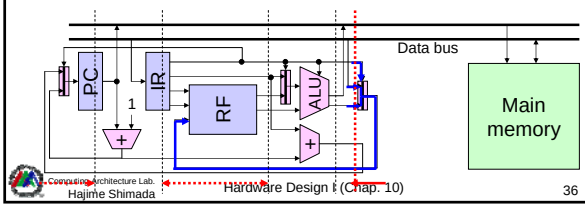
Operation of MA

- Memory access
  - Send generated address to address bus
  - If memory access instruction is load, capture data in data bus
  - If memory access instruction is store, the processor send store data into data bus
- Other instruction: no operation



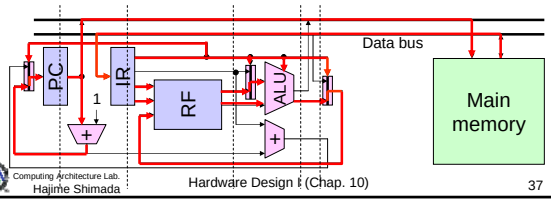
Operation of WB

- Writeback operation result to RF
  - In instructions which creates result
- If branch is taken, write branch target address to PC
  - Coordinated to IF part largely -> See IF part slide



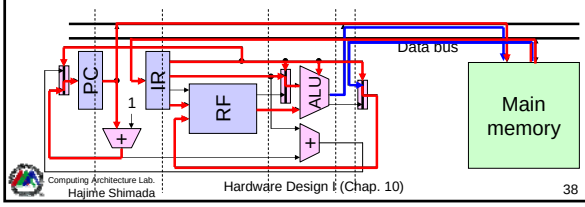
Example execution of “add R8, R4, R5”

1. Send PC to main memory and read instruction into IR
2. Read register by a part of instruction and decode instruction and generate signals
3. Apply arithmetic denoted by a part of instruction
4. Do nothing
5. Writeback arithmetic result to RF and increment PC



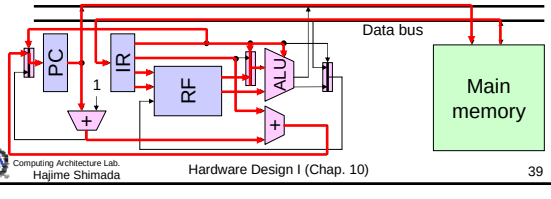
Example execution of “lw R8, 8(R4)”

1. Send PC to main memory and read instruction into IR
2. Read register by a part of instruction, decode instruction and generate signals, and apply sign extension to immediate value
3. Create memory address by adding register and immediate values
4. Send address to memory and read data
5. Writeback read data to RF and increment PC



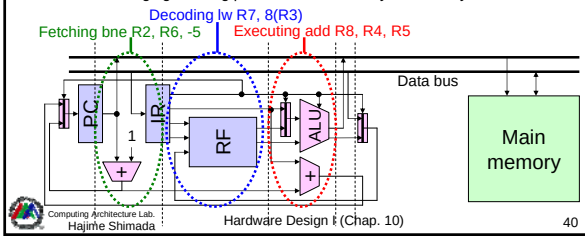
Example execution of “bne R4, R5, -5”

1. Send PC to main memory and read instruction into IR
2. Read register by a part of instruction, decode instruction and generate signals, and apply sign extension to immediate value
3. Check branch condition by arithmetic result of ALU and generate target address with adder
4. Do nothing
5. If condition is taken, writeback target address to PC. Otherwise increment PC



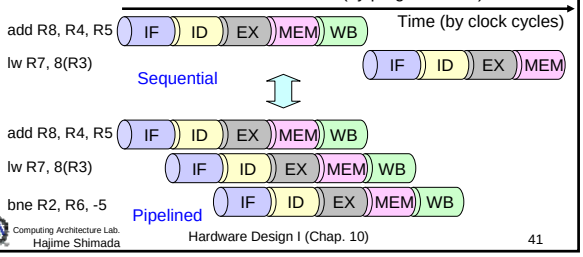
Pipelined processing on processor

- Prior example is sequential execution of 5 parts
- Are there any way to work them simultaneously?
  - Process consecutive instructions in each parts
  - Called “pipelined processing” -> Prof. Yao’s lecture on Jan. 26
    - Imaging creating product on belt conveyer in factory



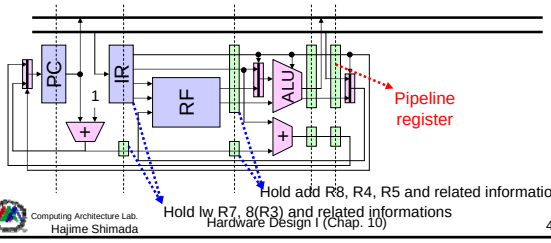
Outlined notation of pipeline

- We usually utilize operation (e.g. IF, ID, ...) denoted in box to represent parallel execution
  - Horizontal axis denotes time (by clock cycles)
  - Vertical axis denotes instruction order (by program order)



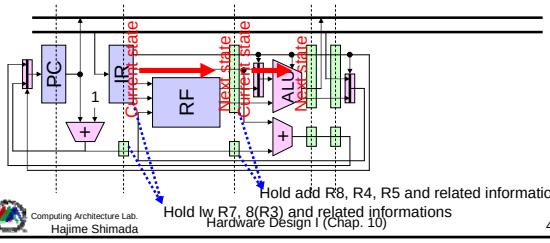
Additional hardware for pipelined processing

- We have to prepare additional FF to keep different instructions
  - Prepare FF between each part called “pipeline register”
  - It keeps not only instructions but also related informations (read register value, arithmetic result, ...)
- Each part is called “pipeline stage” or stage



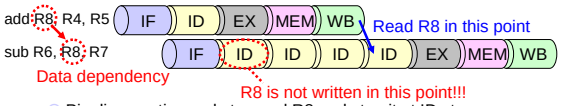
Pipelined processing from sequential circuit viewpoint

- It becomes anomalistic sequential circuit
  - Updated state is written into next FF
- It gives additional constraint to state
  - Caused by relationship between instructions



Pipeline hazard caused by data hazard

- Let's consider following instructions

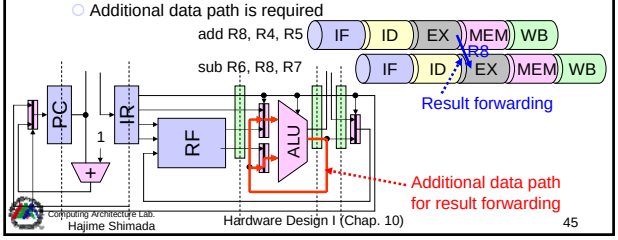


  - Pipeline continuously try read R8 and stop it at ID stage
    - Called "pipeline stall"
  - Called "pipeline hazard": pipeline processing stops with several reasons
- This is a pipeline hazard caused by data dependency
  - Called "data hazard"
  - Data dependency: a relationship that later instruction utilize the result of prior instruction

Data hazard avoidance with result forwarding

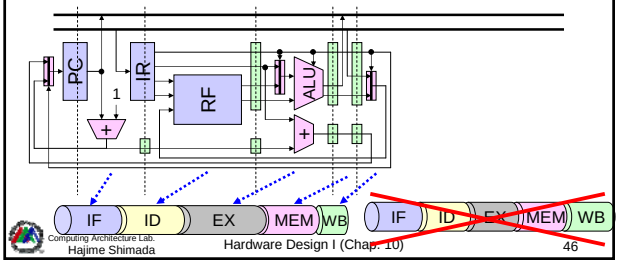
- Pipeline stall is achieved by additional constraint on sequential machine
- Are there any way to avoid it?

  -> Result forwarding: passing value without RF
  - Additional data path is required



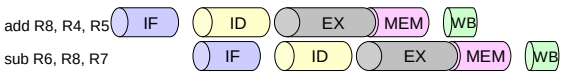
Pipeline and length of logic (1/2)

- Length of (combinational) logic seems the same in outlined figure between stages
- But in practical, it differs



Pipeline and depth of logic (2/2)

- How do we operate those logics?
  - We have to operate them with latest one (which has longest logic) because they works with same clock pulse



- We have to average them as far as possible
- If we consider sequential organization, there's no problem