Digital Computer Logic

Drop Computing Co.

Presented by

6833042821	Chonchanok	Srisongkram
6833032521	Jesadakorn	Netthong
6833071021	Napat	Putsorn
6833098021	Thaweechai	Petchkwan

2110252 Semester 1/2025

Department of Computer Engineering,

Faculty of Engineering, Chulalongkorn University

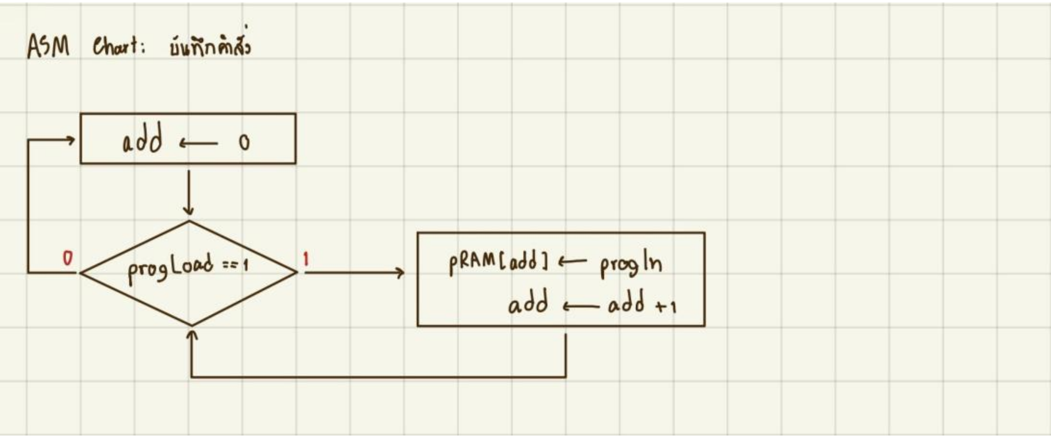
Design concept of CPU

This CPU is created by using the concept of a Multiple Cycle Processor. We divided the work into three phases: **input, execution, and output**.

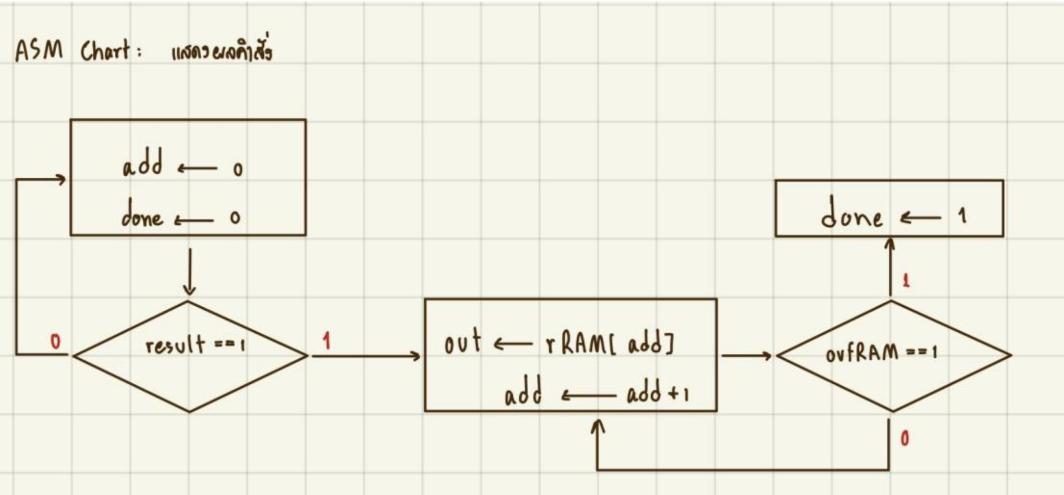
- **Input:** In this phase, the pRAM, which stores the program, will store the input called progIn into the pRAM address when the signal from progLoad is high. It will continue to store the input until the signal from progLoad is low. The size of progIn (input) is 14 bits, divided into the first 6 bits of Opcode, which is an operation number of the operation, and another 8 bits is an operand of the operation.
- **Execution:** After we finish the process in the first phase, all programs in pRAM are ready to be executed. All programs in pRAM will be executed in order from 0-255 when the signal from Start is high. The CPU will fetch the program from pRAM one program at a time. It will then check what this operation needs to do and perform that operation. After finishing the current program, it will fetch and check the next program. The CPU will stop all the processes when it finds the stop operation.
- **Output:** The result that is stored in rRAM is ready to be shown when the stop operation is found. In order to show the result, the result signal must be high. After the result signal is high, the results in rRAM will be displayed in the output in the order they were stored in rRAM. After all results are displayed, the Done signal will be high until the reset signal is high.

ASM Chart

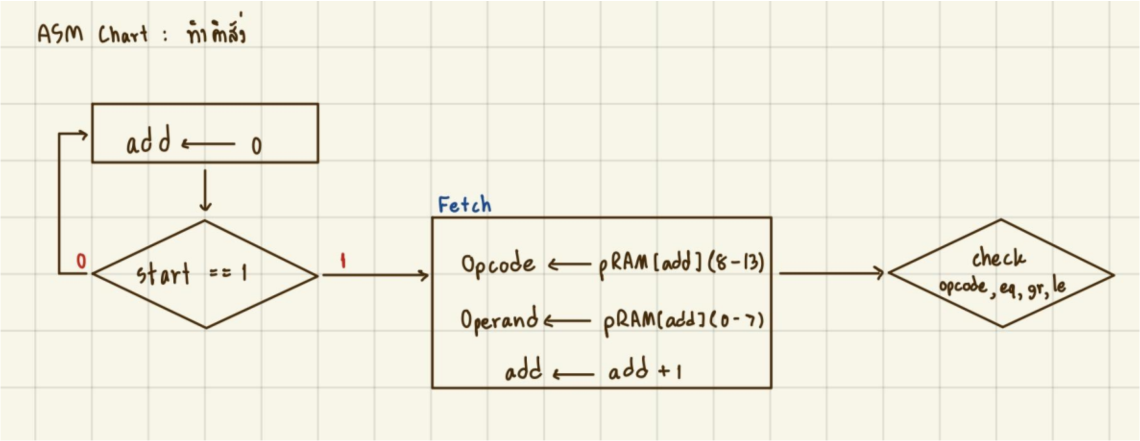
We have 3 ASM Charts, each one is for each phase that is explained in the Design concept of CPU.

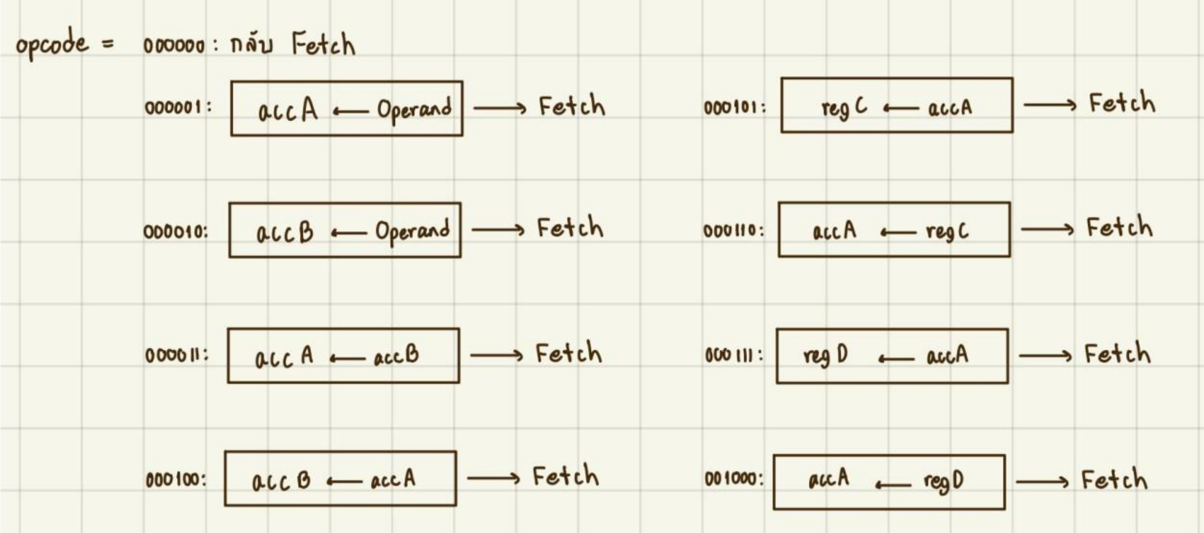


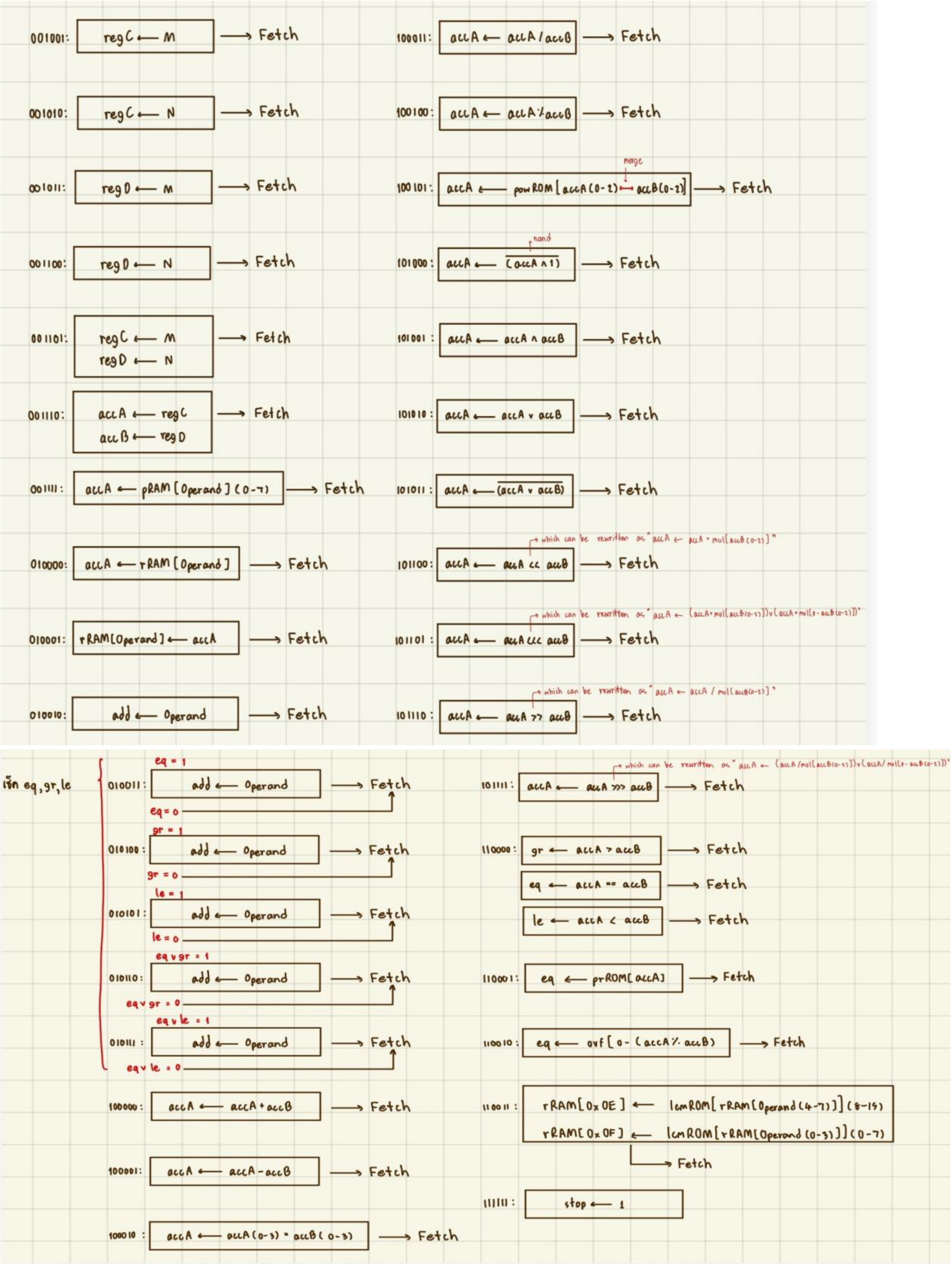
The picture above is the ASM Chart of the Input phase.



The picture above is the ASM Chart of the Output phase.

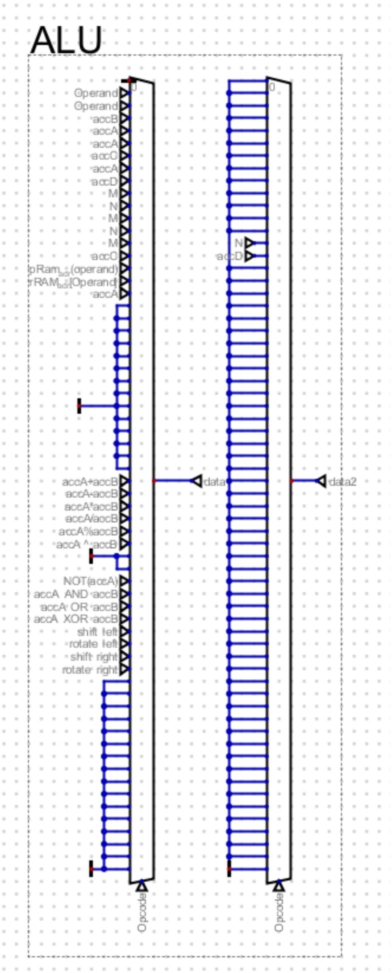


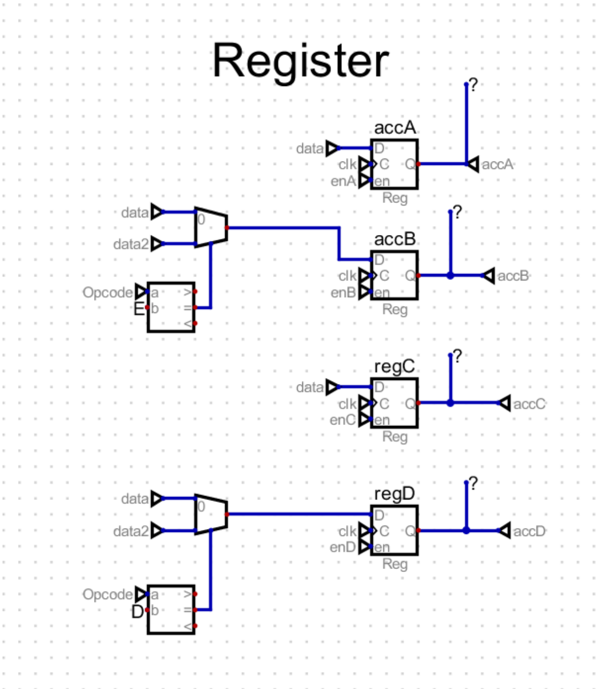

opcode = 000000 : กลับ Fetch
000001 : accA ← Operand → Fetch
000010: accB ← Operand → Fetch
000011: acc A ← accB → Fetch
000100: acc B ← accA → Fetch
000101 : reg C ← accA → Fetch
000110: accA ← reg C → Fetch
000111 : reg D ← accA → Fetch
001000: accA ← reg D → Fetch


001001: regC ← M → Fetch
001010: regC ← N → Fetch
001011: regD ← M → Fetch
001100: regD ← N → Fetch
001101: regC ← M
regD ← N → Fetch
001110: accA ← regC
accB ← regD → Fetch
001111: accA ← pRAM [Operand] (0-7) → Fetch
010000: accA ← rRAM [Operand] → Fetch
010001: rRAM[Operand] ← accA → Fetch
010010: add ← Operand → Fetch
100011: accA ← accA / accB → Fetch
100100: accA ← accA % accB → Fetch
merge
100101: accA ← powROM [accA (0-2) ⟷ accB (0-2)] → Fetch
nand
101000: accA ← (accA ∧ 1) → Fetch
101001: accA ← accA ∧ accB → Fetch
101010: accA ← accA ∨ accB → Fetch
101011: accA ← (accA ∨ accB) → Fetch
which can be rewritten as "accA ← accA * mul[accB (0-2)]"
101100: accA ← accA << accB → Fetch
which can be rewritten as "accA ← (accA * mul[accB(0-2)]) ∨ (accA * mul[8 - accB(0-2)])"
101101: accA ← accA <<< accB → Fetch
which can be rewritten as "accA ← accA / mul[accB(0-2)]"
101110: accA ← accA >> accB → Fetch
เช็ค eq, gr, le
eq = 1
010011: add ← Operand → Fetch
eq = 0
gr = 1
010100: add ← Operand → Fetch
gr = 0
le = 1
010101: add ← Operand → Fetch
le = 0
eq ∨ gr = 1
010110: add ← Operand → Fetch
eq ∨ gr = 0
eq ∨ le = 1
010111: add ← Operand → Fetch
eq ∨ le = 0
100000: accA ← accA + accB → Fetch
100001: accA ← accA - accB → Fetch
100010: accA ← accA (0-3) * accB (0-3) → Fetch
which can be rewritten as "accA ← (accA / mul[accB(0-2)]) ∨ (accA / mul[8 - accB(0-2)])"
101111: accA ← accA >>> accB → Fetch
110000: gr ← accA > accB → Fetch
eq ← accA == accB → Fetch
le ← accA < accB → Fetch
110001: eq ← prROM[accA] → Fetch
110010: eq ← ovf [0 - (accA % accB)] → Fetch
110011: rRAM[0x0E] ← lcmROM[rRAM[Operand (4-7)]] (8-15)
rRAM[0x0F] ← lcmROM[rRAM[Operand (0-3)]] (0-7)
Fetch
111111: stop ← 1

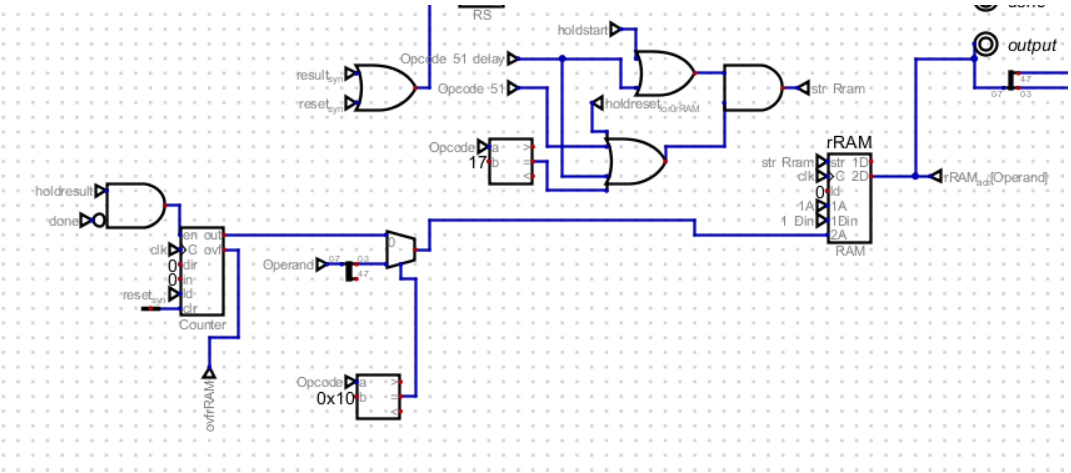
The pictures above are the ASM Chart of the Execution phase

Data Path



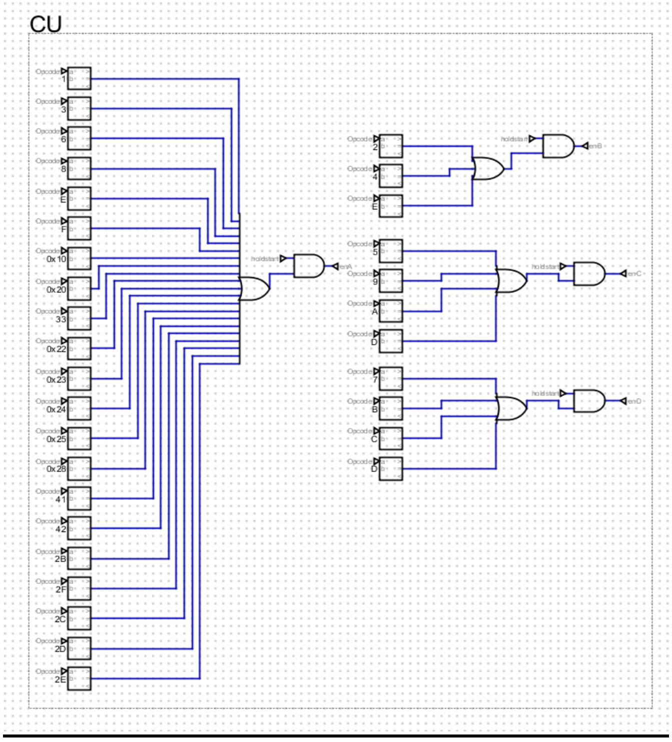


- 2 Mux 64:1* are used to select the data, processed by ALU, to use by other function or store back to register, controlled by Opcode.
(*with some exception for example, jump function, comparison that don't need to store the data)

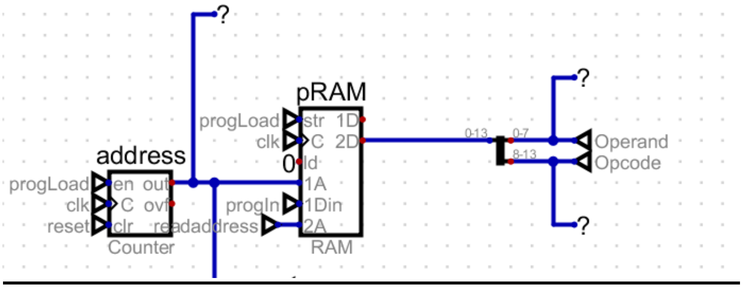


-the rRam is used for store the value from calculation or operation that Opcode has ordered, by storing in address from 0x00 to 0x0F

Control Unit



-this part of control unit will control what accumulator or register that need to store the value by comparing the received Opcode to the set of Opcode to allow register to store data



-This part of the control unit contains the Program Counter ("address") , IR, and pRam, which are used to minimize stall time and prevent data hazards. Which Program Counter needs to use as little time as possible before fetching the next operation?

- Program Counter (PC), which is "address", used to fetch both Opcode and Operand in the pRam to send out of the 2D port
- IR is used to separate Opcode and Operand from the incoming code of progIn
- pRam is used to store the operations before the CPU fetches the operations to run