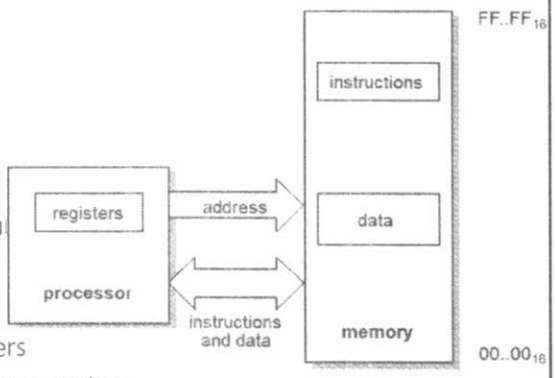
Project
mu0 processor

Soongsil University
Chanho Lee

Processor architecture and organization

- Computer system
  - ▶ Processor + Memory system
- Computer architecture
  - ▶ The user's view of the computer
  - ▶ Instruction set, visible registers, ...
- Computer organization
  - ▶ The user-invisible implementation of the architectu
  - ▶ Pipeline structure, transparent cache, ...
- The stored-program computer
  - ▶ Employed by all modern general-purpose computers
  - ▶ It keeps its instructions and data in the same memory system



Processor architecture and organization

- Processor
  - ▶ A finite-state automaton that executes instructions held in a memory
  - ▶ State of the system
    - Values held in the memory
    - Values held in registers within the processor itself
  - ▶ Instruction
    - It defines a particular way the total state should change
    - It defines which instruction should be executed next

MU0 – a simple processor

- MU0
  - ▶ A design developed only for teaching at the University of Manchester
  - ▶ Illustrate the basic principles of processor design
  - ▶ It is a very simple processor
    - It would not make a good target for a high-level language compiler
  - ▶ Non-pipelined operation
- MU0 architecture
  - ▶ 16 bit machine: 2 byte/word
  - ▶ 12 bit address space
    - It can address up to 8Kbyte
    - 4K individually addressable 16-bit locations

MU0 – a simple processor

- Components
  - ▶ Program counter (PC)
    - Holds the address of the current instruction
  - ▶ Accumulator (ACC)
    - Holds a data value while it is worked upon
  - ▶ Arithmetic-logic unit (ALU)
    - Performs a number of operations on binary operands, such as add, subtract, increment, ...
  - ▶ Instruction register (IR)
    - Holds the current instruction while it is executed
  - ▶ Instruction decode and control logic
    - Employs the above components to achieve the desired results from each instruction

MU0 – a simple processor

- Instruction set
  - ▶ Instruction format

4bit	12bit
opcode	S

  - ▶ Instructions

Instruction	Opcode	Effect
LDA S	0000	ACC := mem_{16}[S]
STO S	0001	mem_{16}[S] := ACC
ADD S	0010	ACC := ACC + mem_{16}[S]
SUB S	0011	ACC := ACC - mem_{16}[S]
JMP S	0100	PC := S
JGE S	0101	If ACC >= 0 PC := S
JNE S	0110	If ACC != 0 PC := S
STP	0111	stop

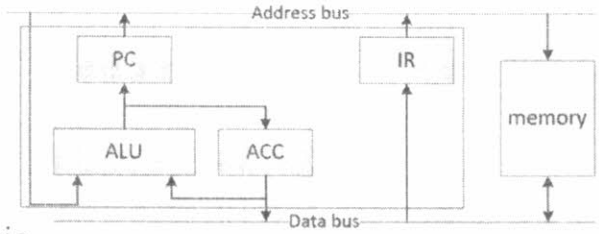
MU0 has spaces left in the instruction space which allow future expansion of the instruction set

MU0 – a simple processor

- Logic design
  - ▶ Datapath
    - All the components carrying, storing or processing many bits in parallel
    - Including the accumulator, program counter, ALU and instruction register
    - Designed using a register transfer level design style based on register, multiplexers, and so on.
  - ▶ Control logic
    - everything that does not fit comfortably into datapath
    - Control datapath components
    - Designed using a finite state machine (FSM) approach

MU0 – a simple processor

- Datapath design
  - ▶ There are many ways to connect the components
  - ▶ We will follow the principle that the memory will be the limiting factor
  - ▶ A memory access will always take a clock cycle
    - Each instruction takes exactly the number of clock cycles defined by the number of memory accesses it must take



- Control logic
  - ▶ Control datapath components
  - ▶ Designed using a finite state machine (FSM) approach

MU0 – a simple processor

- Datapath operation
  - ▶ EX state: Execute
    - The address in the instruction register is issued
    - Either an operand is read from memory, combined with the accumulator in the ALU and written back into the accumulator
    - or the accumulator is stored out to memory
  - ▶ IF state: fetch the next instruction
    - Either PC or the address in the instruction register is issued to fetch the next instruction
    - Address is incremented in the ALU and value saved into the PC

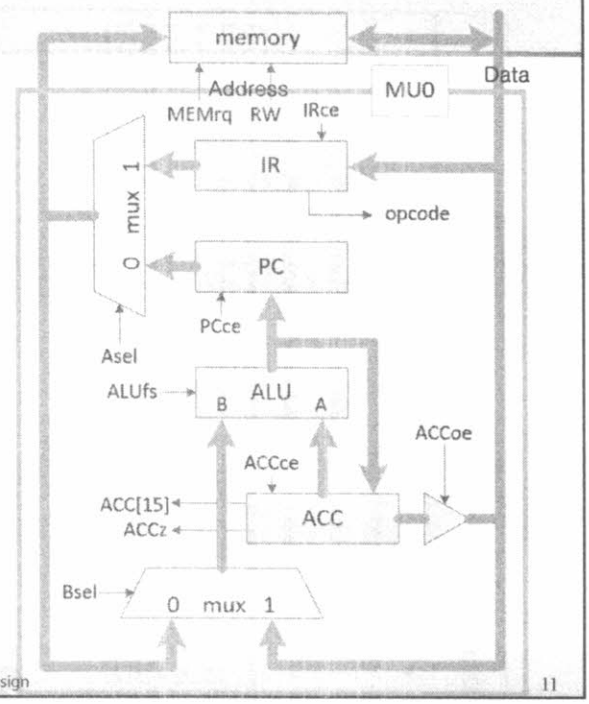
MU0 – a simple processor

- Datapath operation
  - ▶ each instruction starts when it has arrived in the instruction register

Instruction	Opcode	Effect	
LDA S	0000	ACC := mem_{16}[S]	EX, IF
STO S	0001	mem_{16}[S] := ACC	EX, IF
ADD S	0010	ACC := ACC + mem_{16}[S]	EX, IF
SUB S	0011	ACC := ACC - mem_{16}[S]	EX, IF
JMP S	0100	PC := S	IF
JGE S	0101	if ACC >= 0 PC := S	IF
JNE S	0110	if ACC !=0 PC := S	IF
STP	0111	stop	IF

MU0 – a simple processor

- RTL design
  - ▶ Determine the control signals that are required to cause the datapath to carry out the full set of operations
  - ▶ Assume that all the registers change state on the falling edge of the input clock
- Initialization
  - ▶ The processor must start in a known state
  - ▶ Reset input to start executing instructions from a known address; here it is 0x000

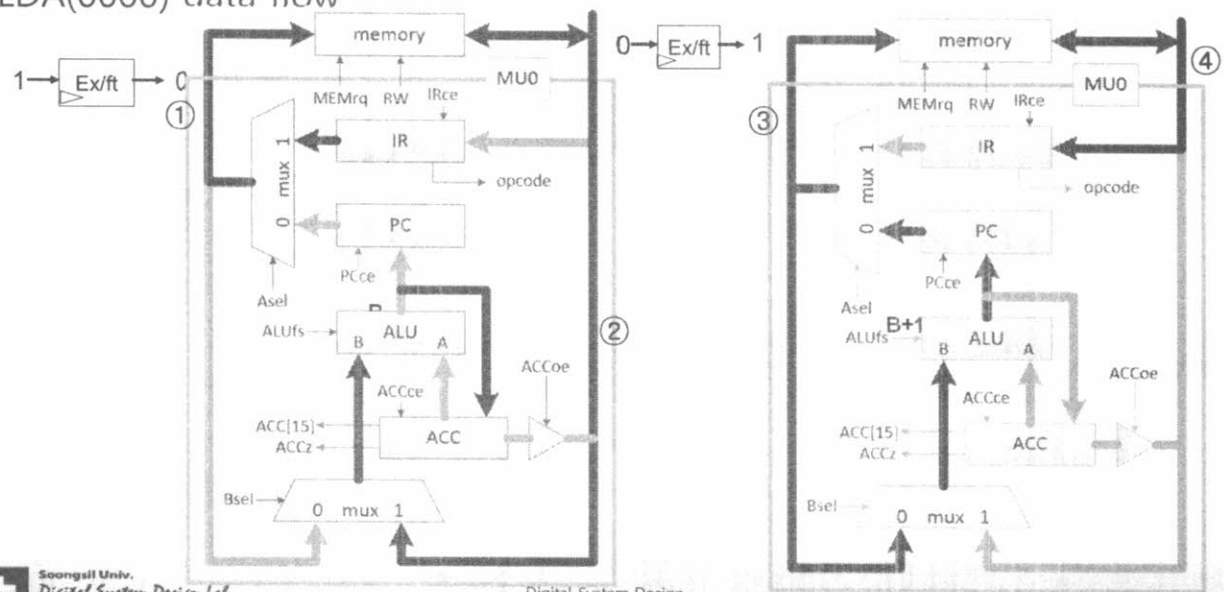


MU0 – a simple processor

- Datapath control signals
  - ▶ Asel: A-MUX(address) select control
  - ▶ Bsel: B-MUX(ALU's B operand) select control
  - ▶ ACCce: Accumulator change enable
  - ▶ PCce: PC change enable
  - ▶ IRce: Instruction register change enable
  - ▶ ACCoe: Accumulator output enable (tri-state buffer control)
  - ▶ ALUfs: ALU function select
- Memory interface signals
  - ▶ MEMrq: memory request
  - ▶ RnW: read(1)/write(0)
- Control logic state
  - ▶ Ex/ft: execute(0)/fetch(1)
- Control logic inputs from the datapath
  - ▶ opcode: operation code
  - ▶ ACC[15]: indicate that the value saved in ACC is negative
  - ▶ ACCz: indicate that the value saved in ACC is zero

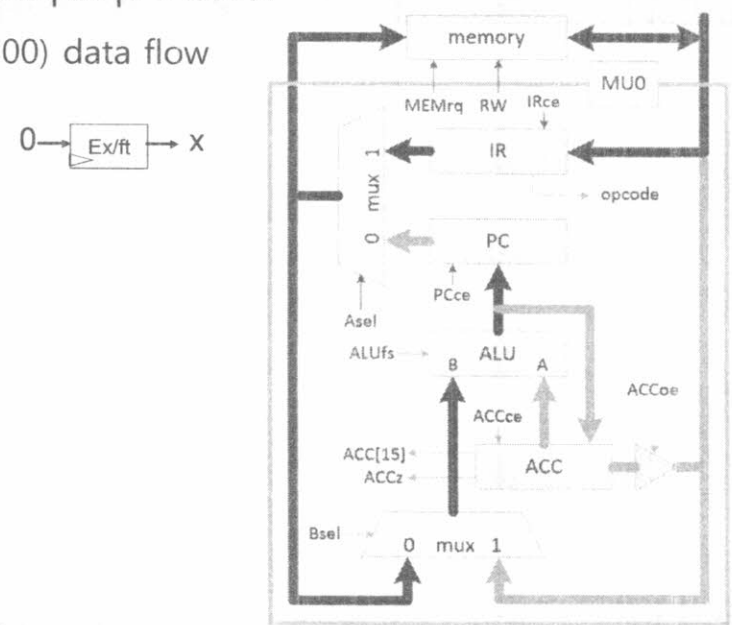
MU0 – a simple processor

LDA(0000) data flow



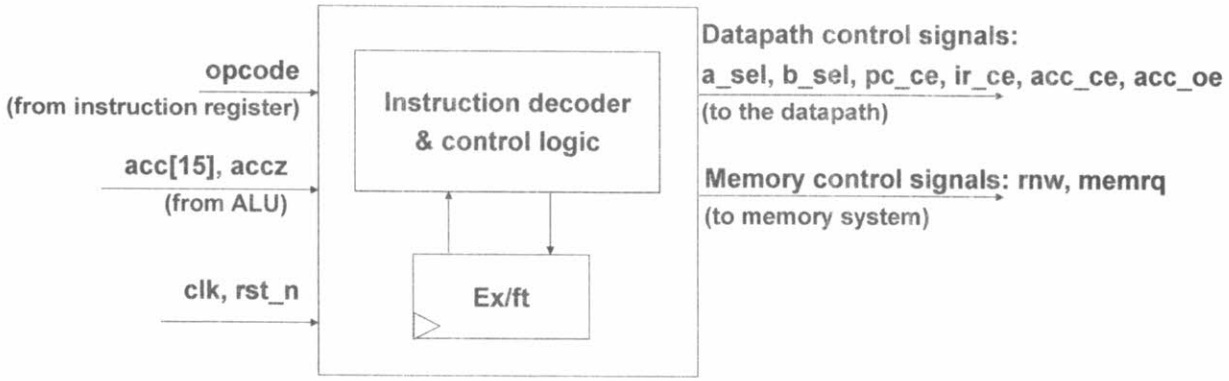
MU0 – a simple processor

- JMP(0100) data flow



MU0 – a simple processor

- Control logic



MU0 – a simple processor

- *Control logic*

Inputs						Outputs									
	Opcode	Ex/ft	ACC15				Bsel		PCce		ACCoe		MEMrq	Ex/ft	
Instruction		Reset	ACCz			Asel	ACCce		IRce		ALUfs		RnW		
Reset	xxxx	1	x	x	x	0	0	1	1	1	0	=0	1	1	0
LDA S	0000	0	0	x	x	1	1	1	0	0	0	=B	1	1	1
	0000	0	1	x	x	0	0	0	1	1	0	B+1	1	1	0
STO S	0001	0	0	x	x	1	x	0	0	0	1	x	1	0	1
	0001	0	1	x	x	0	0	0	1	1	0	B+1	1	1	0
ADD S	0010	0	0	x	x	1	1	1	0	0	0	A+B	1	1	1
	0010	0	1	x	x	0	0	0	1	1	0	B+1	1	1	0
SUB S	0011	0	0	x	x	1	1	1	0	0	0	A-B	1	1	1
	0011	0	1	x	x	0	0	0	1	1	0	B+1	1	1	0
JMP S	0100	0	x	x	x	1	0	0	1	1	0	B+1	1	1	0
JGE S	0101	0	x	x	0	1	0	0	1	1	0	B+1	1	1	0
	0101	0	x	x	1	0	0	0	1	1	0	B+1	1	1	0
JNE S	0110	0	x	0	x	1	0	0	1	1	0	B+1	1	1	0
	0110	0	x	1	x	0	0	0	1	1	0	B+1	1	1	0
STOP	0111	0	x	x	x	1	x	0	0	0	0	x	0	1	0

Present state

Next state

MU0 – a simple processor

- ALU design

Operation	ALUfs
0	0
A+B	1
A-B	2
B	3
B+1	4

Project (required)

- Task: obtain $\sum_{i=S}^{N} i$ (N>=S)
- Write an Verilog code for the original MU0 processor
  - ▶ Obtain the waveform for the operation above
  - ▶ Memory may be defined as the size of 32 x 16 bits for the instructions and data
  - ▶ Operation of all instructions must be verified.
- Write an assembly code for the task using the original instructions. Count the number of cycles for the operation.
  - ▶ The above code must be verified by ModelSim simulation

Project (Optional; up to +50% points)

- Task: obtain $\sum_{i=S}^{N} i$ (N>=S)
- Write an assembly code for the task including the extended instructions.
  - ▶ Extend instructions of MU0 processor
  - ▶ The architecture of MU0 processor need to be modified for the extended instructions
  - ▶ E.g. register, arithmetic unit, or other blocks may be added
  - ▶ The results should be obtained with the minimum number of executions including extended instructions; less number of execution than that using the original MU0 processor
- Write an Verilog code for the extended MU0 processor
  - ▶ Obtain the waveform for the operations above
  - ▶ Memory may be defined as the size of 32 x 32 bits for the instructions and data

Project report and presentation

- Introduction
  - ▶ Including roles of each team member
- Processor block diagram
  - ▶ Components and detailed signals
- Verification of instructions
  - ▶ Simulation plan
  - ▶ ModelSim simulation results and analysis
- Verification of the task
  - ▶ Simulation plan
  - ▶ Analysis of assembly code and the number of operation cycles
  - ▶ ModelSim simulation results and analysis
- Synthesis results
- Summary

Project

- Assembly code example for the task using the original instructions.

```
1   LDA S          //acc:=mem16[S]
2   STO sum        //mem16[sum]:=acc(S)          sum:=S;
3   STO i          //mem16[i] := acc(S)
4   SUB N          //acc(S):=acc(S)-mem16[N]
5   JNE loop1      //if acc != 0 pc:=loop1       if (S != N)
6   STP            //stop
loop1:
7   LDA i          //acc:=mem16[i]               for (i:=S+1;i<=N;i:=i+1)
8   ADD v1         //acc(i):= acc(i)+mem16[v1]
9   STO i          //mem16[i] := acc(i+1)
10  ADD sum        //acc(sum):=acc(i)+mem16[sum] sum:=sum+i;
11  STO sum        //mem16[sum] := acc(sum)
12  LDA i          //acc:=mem16[i]
13  SUB N          //acc(i):=acc(i)-mem16[N]
14  JNE loop1      //if acc != 0 pc:=loop1
15  STP            //stop
```