Lecture 01

Introduction to Processor Design (1)

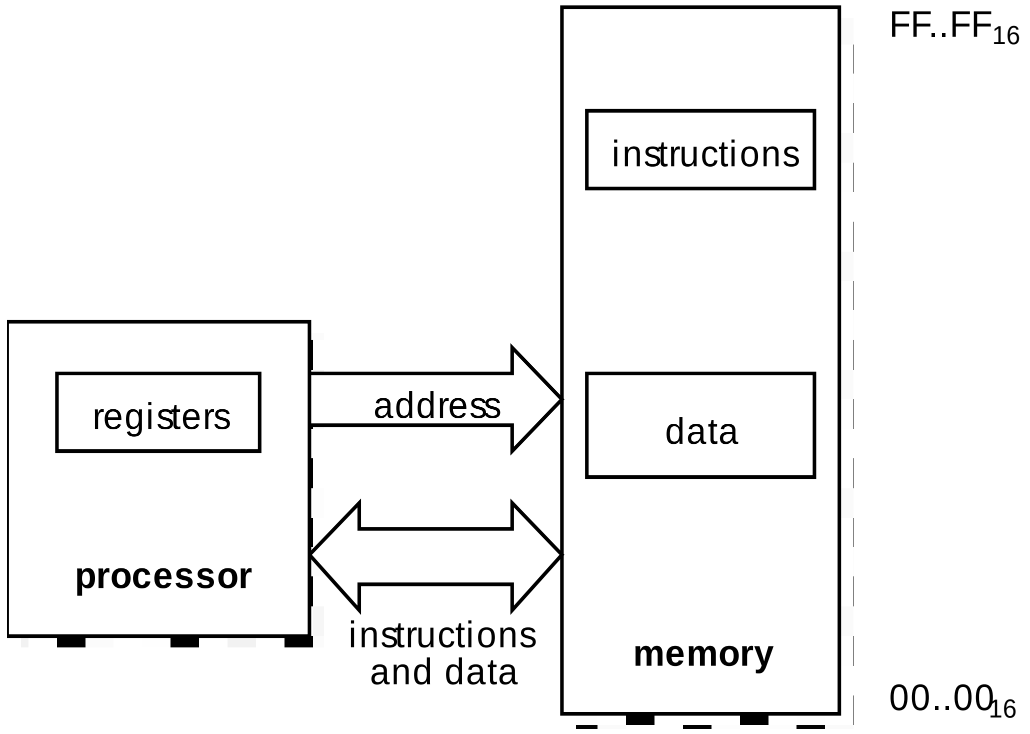
Computer Architecture vs. Organization

- Computer Architecture
  - Computer architecture describes the user's view of the computer. The instruction set, visible registers, memory management table structure and exception handling modal are all part of the architecture.

- Computer Organization
  - Computer organization describes the user-invisible implementation of the architecture. The pipeline structure, transparent cache, table-walking hardware and TLB are all aspects of the organization.

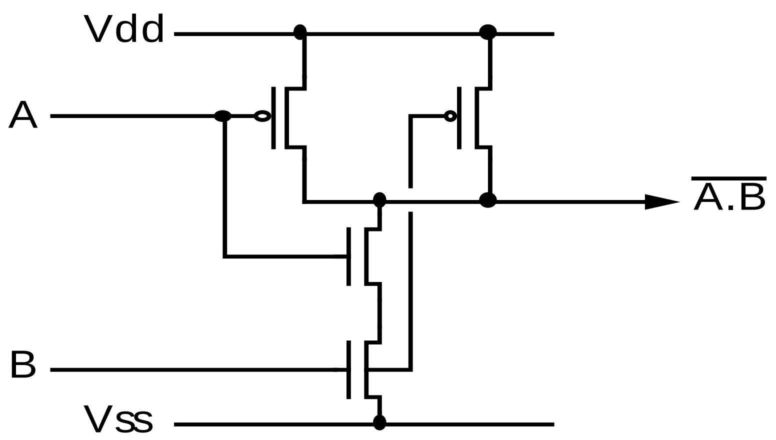
What is processor?

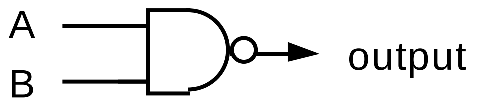
- A general-purpose processor is a finite-state automaton that executes instr. held in memory.
- The state of the system is defined by the values held in the memory locations together with the values held in certain registers within the processor itself.



Abstraction in hardware design

- A typical hierarchy of abstraction at the hardware level might be:
  - Transistors
  - Logic gates, memory cells, special circuits;
  - Single-bit adders, multiplexers, decoders, flip-flops;
  - Word-wide adders, multiplexers, decoders, registers, buses;
  - ALUs, barrel shifters, register banks, memory blocks;
  - Processor, cache and memory management organizations;
  - Processors, peripheral cells, cache memories, memory management units;
  - Integrated system chip;
  - Printed circuit boards;
  - Mobile telephones, PCs, engine controllers.





Logic symbol

A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

Truth table

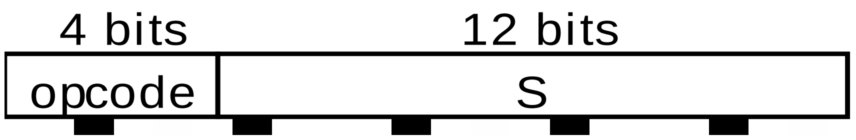
MU0 – A simple processor

- A simple processor includes a few basic components:
  - A program counter (PC)
  - A single register called an accumulator (ACC)
  - An arithmetic-logic unit (ALU)
  - An instruction register (IR)
  - Instruction decode and control logic

- Manchester-designed machines are often referred to by the names MU*n* for 1<=*n*<=6, so this simple machine is known as MU0.

The MU0 instruction format

- The MU0 instruction formant



- MU0 is a 16-bit machine with a 12-bit address space, so it can address up to 8Kbytes of memory arranged as 4096 individually addressable 16-bit locations.

- Instructions are 16 bits long, with a 4-bit operation code (or opcode) and a 12-bit address field (S).

The MU0 instruction Set

- The MU0 instruction set

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

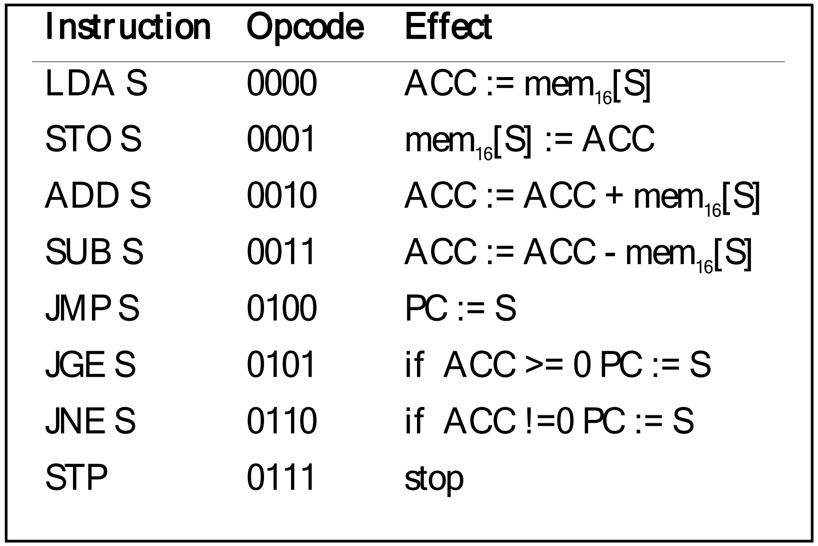
MU0 logic design

- The datapath
  - All the components carrying, storing or processing many bits in parallel will be considered part of the datapath, including the ACC, PC, ALU and IR. For these components we will use a register transfer level (RTL) design style based on registers, multiplexers, and so on.
- The control logic
  - Everything that does not fit comfortably into the datapath will be considered part of the control logic and will be design using a finite state machine (FSM) approach.

Datapath Design

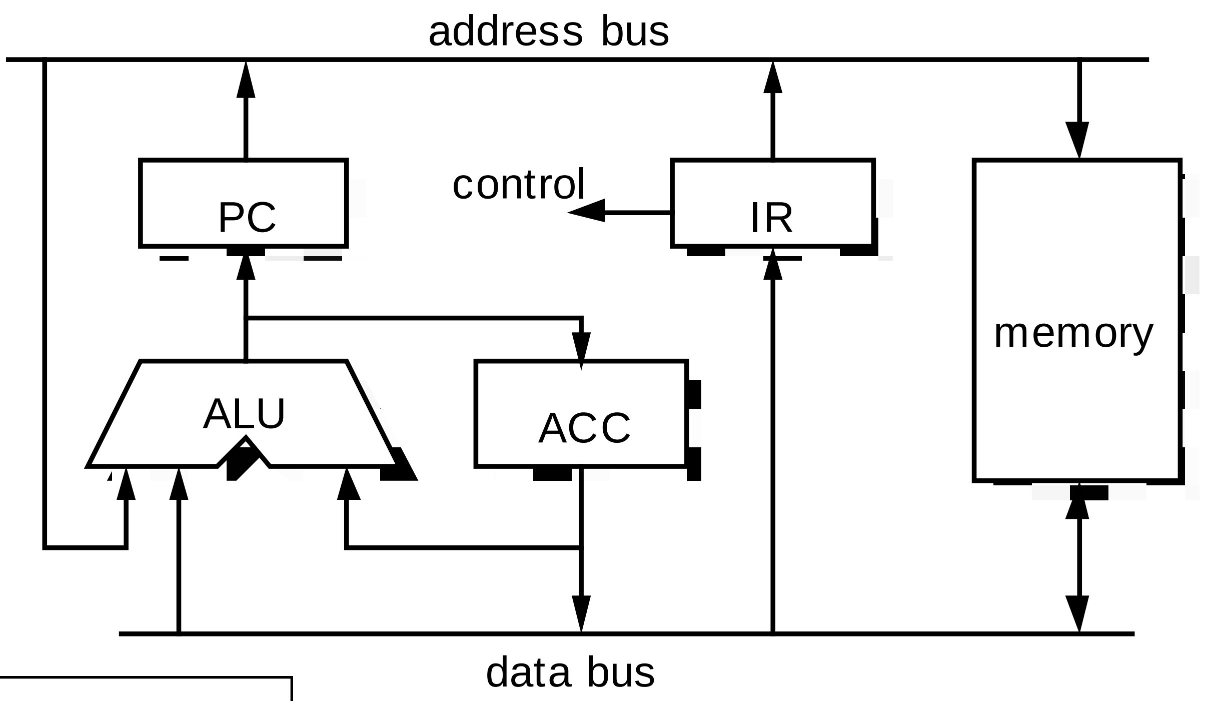
- Each instruction takes exactly the number of clock cycles defined by the number of memory access is must make
  - The first four instructions each require two memory accesses.
  - The last four only require one cycle.

- Readers who might expect to see a dedicated PC incrementer in this datapath should note that all instructions that do not change the PC take two cycle, so the main ALU is available during one of these cycles to increment the PC.



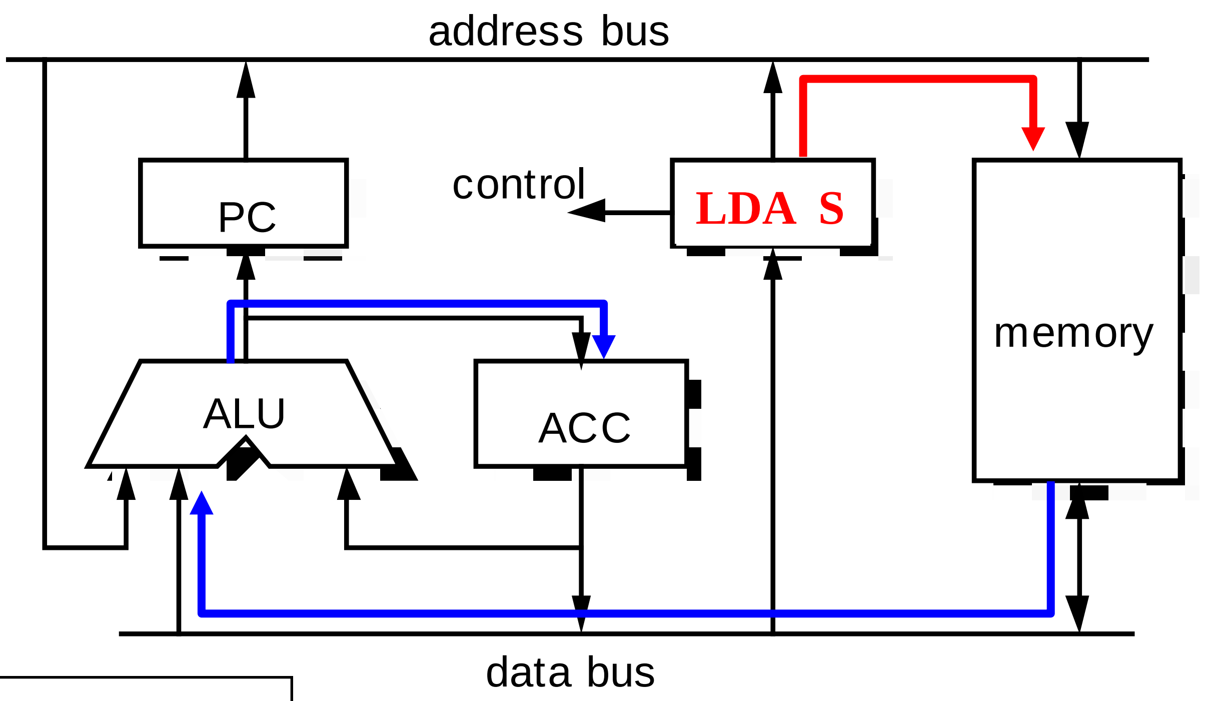
Instruction	Opcode	Effect
LDA S	0000	ACC := mem_{16}[S]
STO S	0001	mem_{16}[S] := ACC
ADD S	0010	ACC := ACC + mem_{16}[S]
SUB S	0011	ACC := ACC - mem_{16}[S]
JMP S	0100	PC := S
JGE S	0101	if ACC >= 0 PC := S
JNE S	0110	if ACC != 0 PC := S
STP	0111	stop

MU0 datapath example



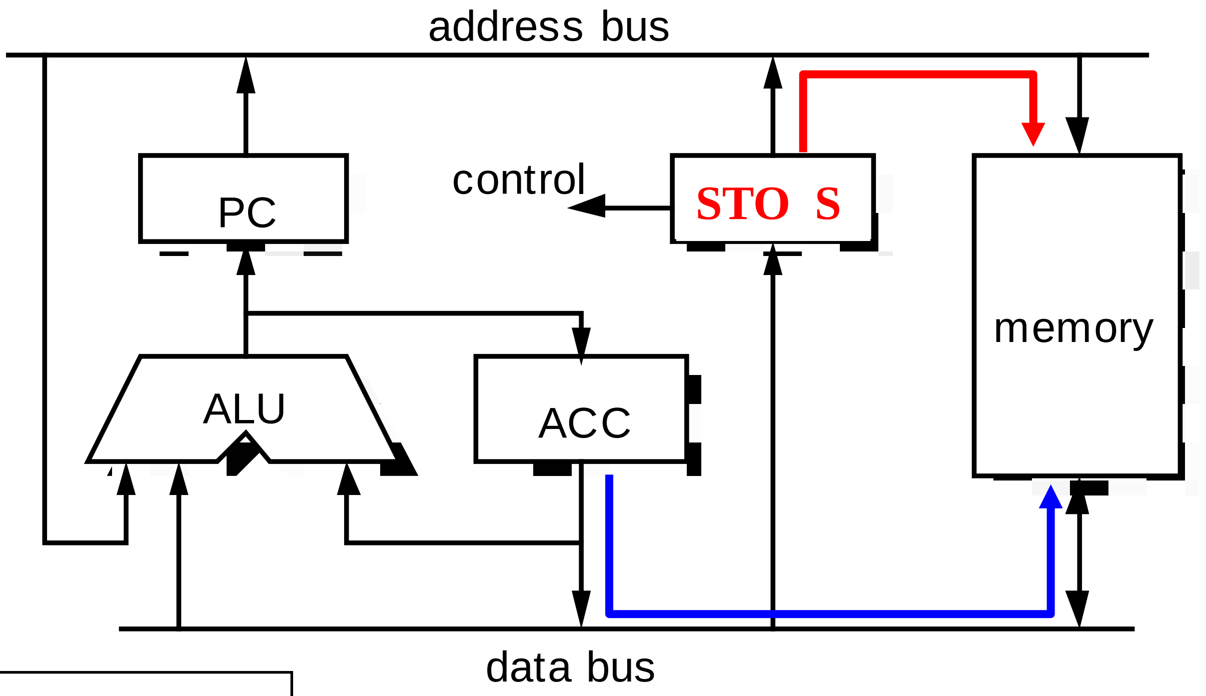
Instruction	Opcode	Effect
LDA S	0000	ACC := mem_{16}[S]
STO S	0001	mem_{16}[S] := ACC
ADD S	0010	ACC := ACC + mem_{16}[S]
SUB S	0011	ACC := ACC - mem_{16}[S]
JMP S	0100	PC := S
JGE S	0101	if ACC >= 0 PC := S
JNE S	0110	if ACC !=0 PC := S
STP	0111	stop

MU0 datapath example (1)



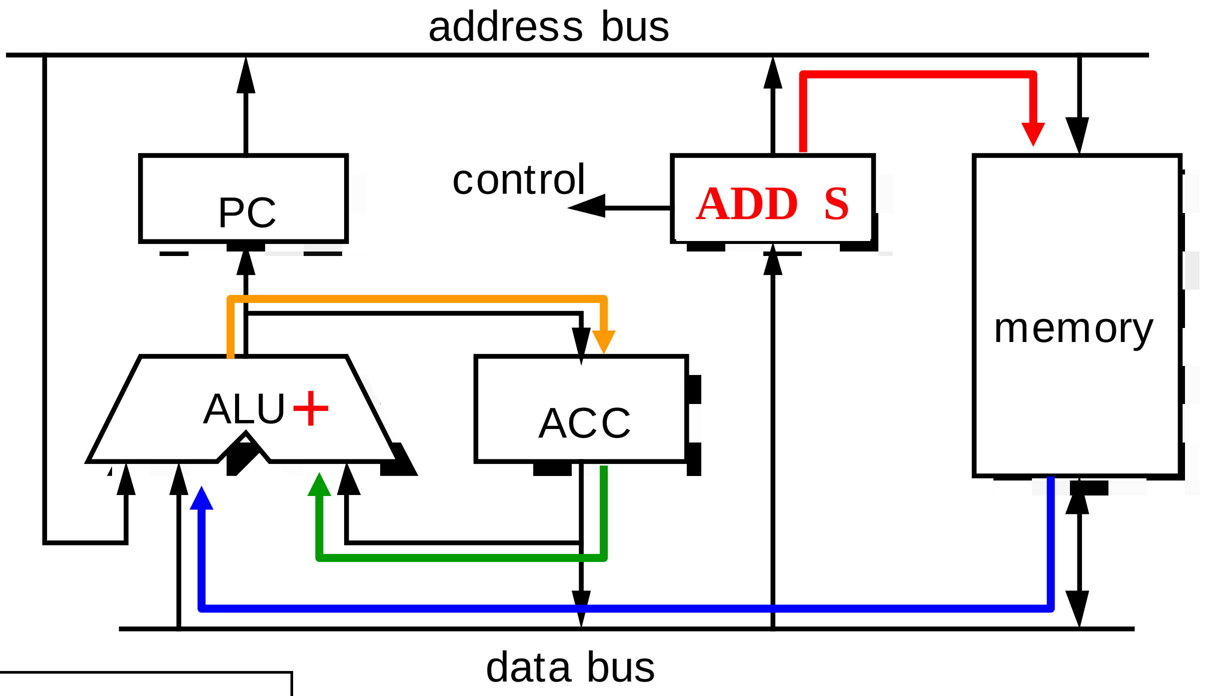
Instruction	Opcode	Effect
LDA S	0000	ACC := mem_{16}[S]
STO S	0001	mem_{16}[S] := ACC
ADD S	0010	ACC := ACC + mem_{16}[S]
SUB S	0011	ACC := ACC - mem_{16}[S]
JMP S	0100	PC := S
JGE S	0101	if ACC >= 0 PC := S
JNE S	0110	if ACC !=0 PC := S
STP	0111	stop

MU0 datapath example (2)



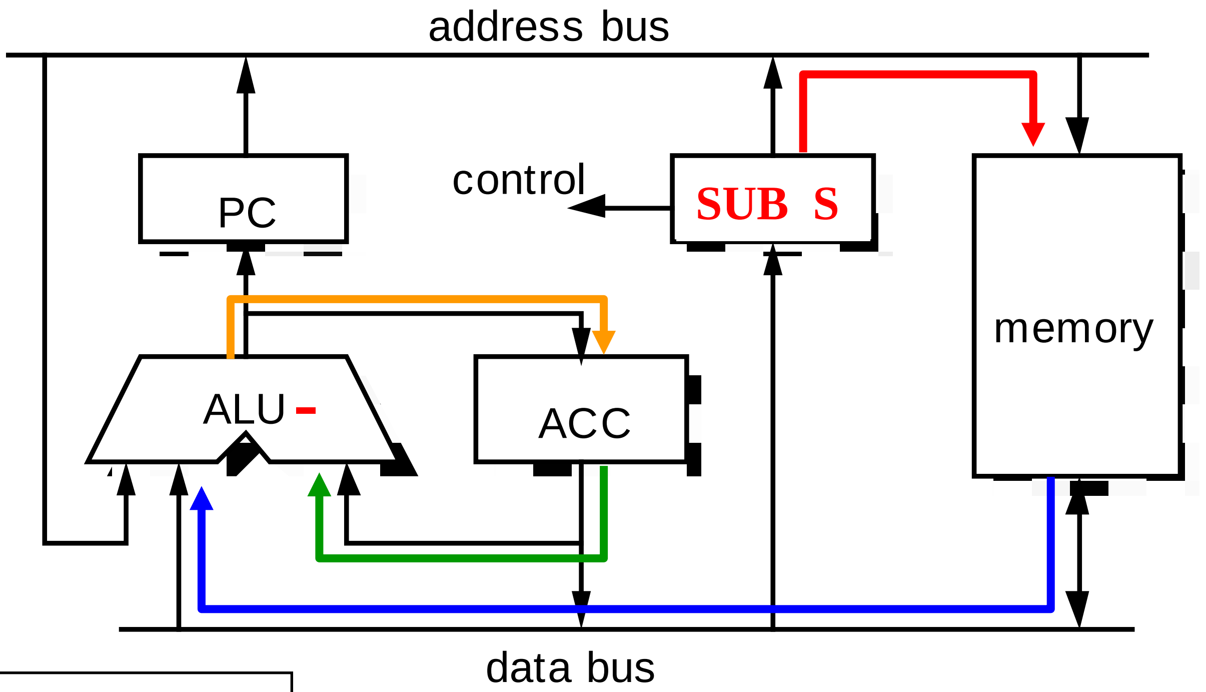
Instruction	Opcode	Effect
LDA S	0000	ACC := mem_{16}[S]
STO S	0001	mem_{16}[S] := ACC
ADD S	0010	ACC := ACC + mem_{16}[S]
SUB S	0011	ACC := ACC - mem_{16}[S]
JMP S	0100	PC := S
JGE S	0101	if ACC >= 0 PC := S
JNE S	0110	if ACC !=0 PC := S
STP	0111	stop

MU0 datapath example (3)



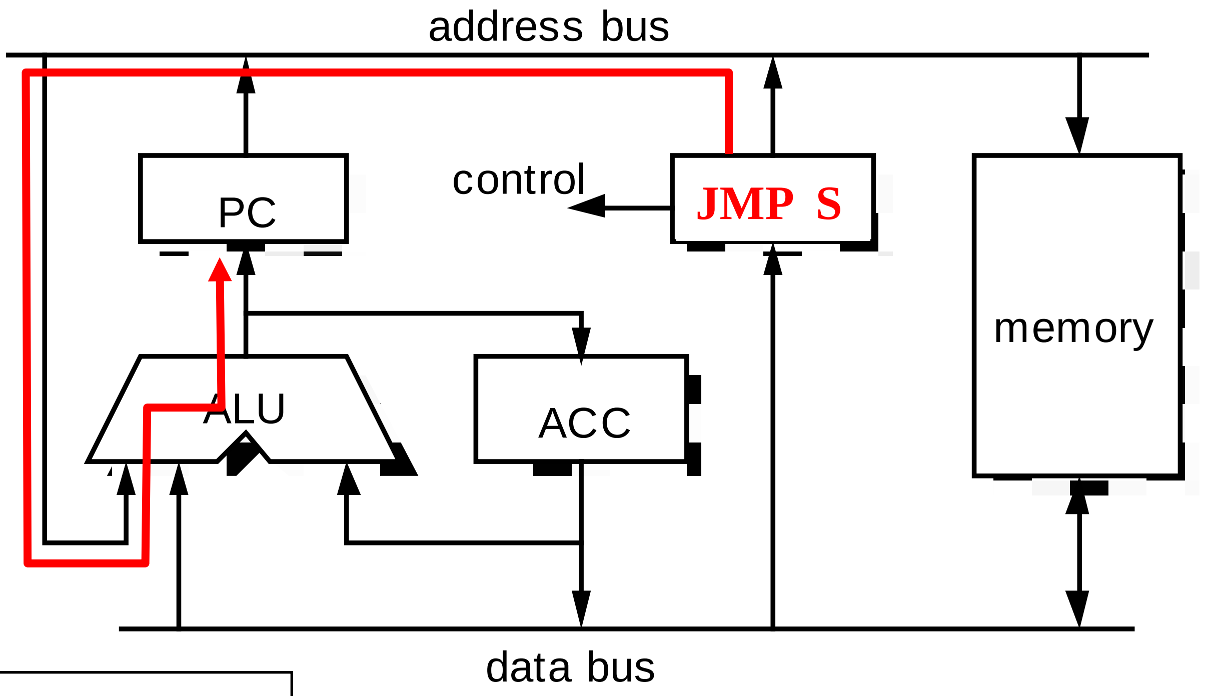
Instruction	Opcode	Effect
LDA S	0000	ACC := mem_{16}[S]
STO S	0001	mem_{16}[S] := ACC
ADD S	0010	ACC := ACC + mem_{16}[S]
SUB S	0011	ACC := ACC - mem_{16}[S]
JMP S	0100	PC := S
JGE S	0101	if ACC >= 0 PC := S
JNE S	0110	if ACC !=0 PC := S
STP	0111	stop

MU0 datapath example (4)



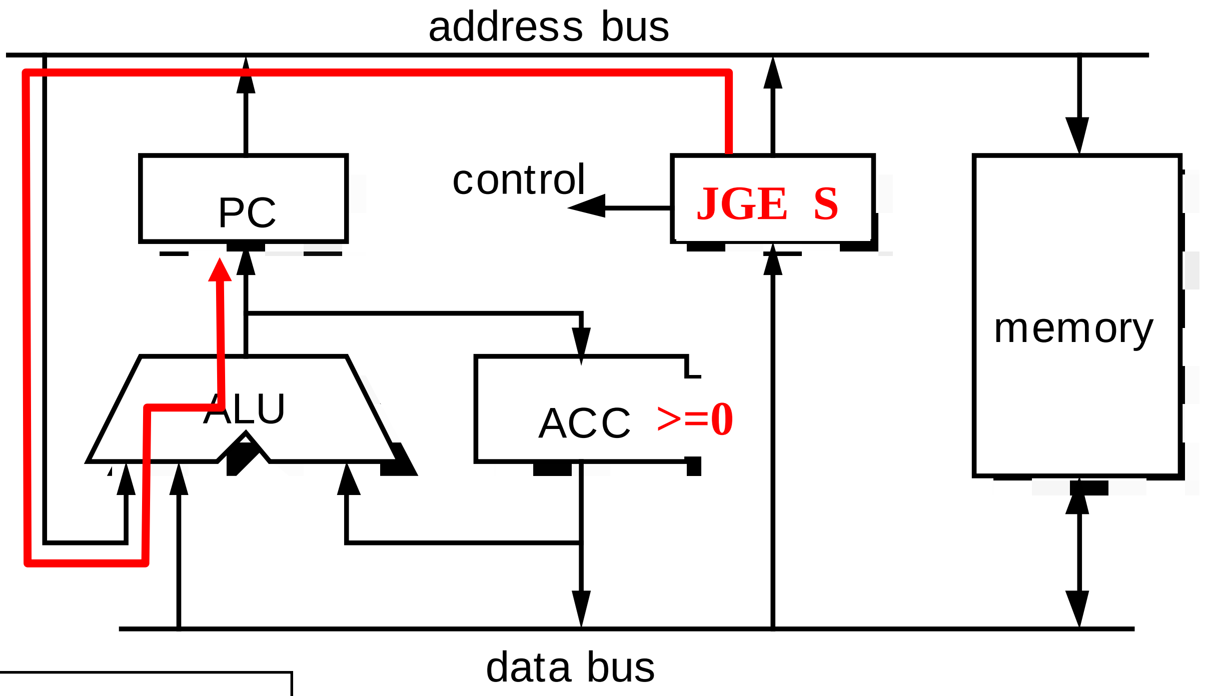
Instruction	Opcode	Effect
LDA S	0000	ACC := mem_{16}[S]
STO S	0001	mem_{16}[S] := ACC
ADD S	0010	ACC := ACC + mem_{16}[S]
SUB S	0011	ACC := ACC - mem_{16}[S]
JMP S	0100	PC := S
JGE S	0101	if ACC >= 0 PC := S
JNE S	0110	if ACC !=0 PC := S
STP	0111	stop

MU0 datapath example (5)



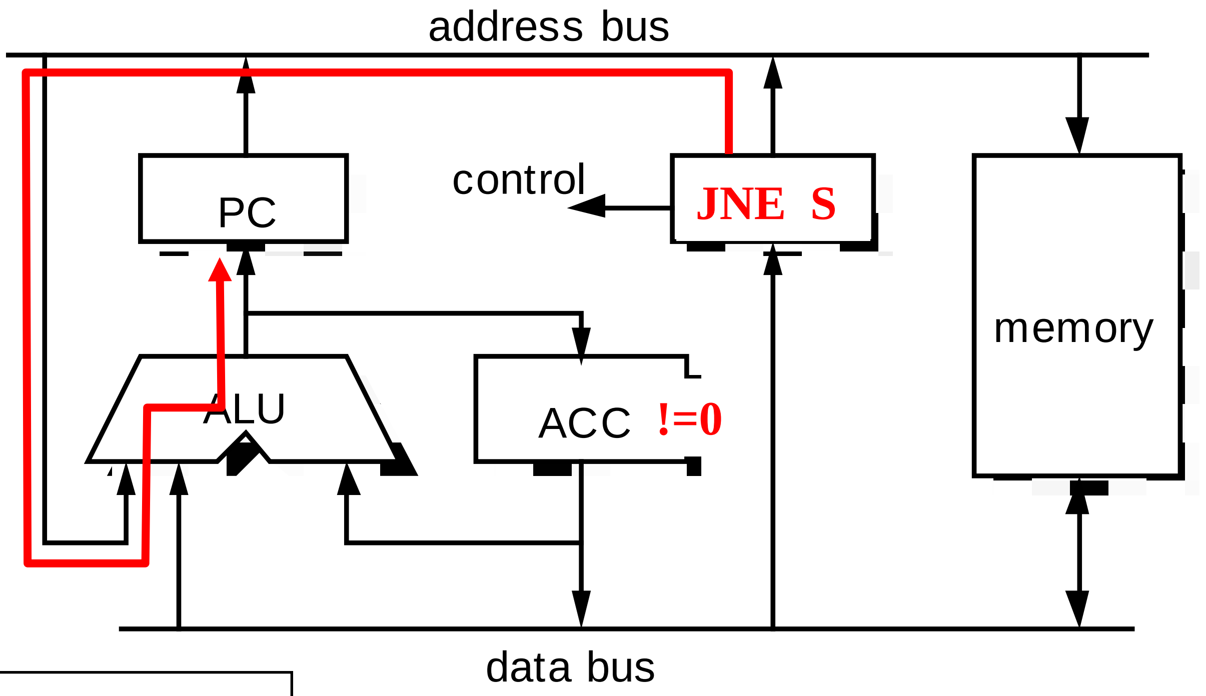
Instruction	Opcode	Effect
LDA S	0000	ACC := mem_{16}[S]
STO S	0001	mem_{16}[S] := ACC
ADD S	0010	ACC := ACC + mem_{16}[S]
SUB S	0011	ACC := ACC - mem_{16}[S]
JMP S	0100	PC := S
JGE S	0101	if ACC >= 0 PC := S
JNE S	0110	if ACC !=0 PC := S
STP	0111	stop

MU0 datapath example (6)



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 datapath example (6)



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

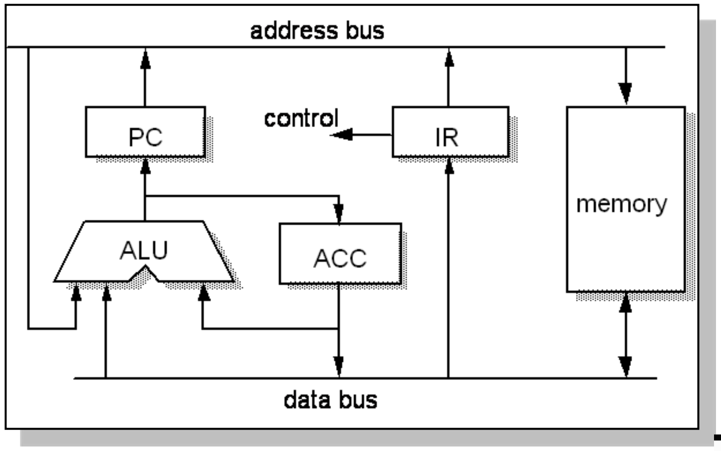
Datapath Design

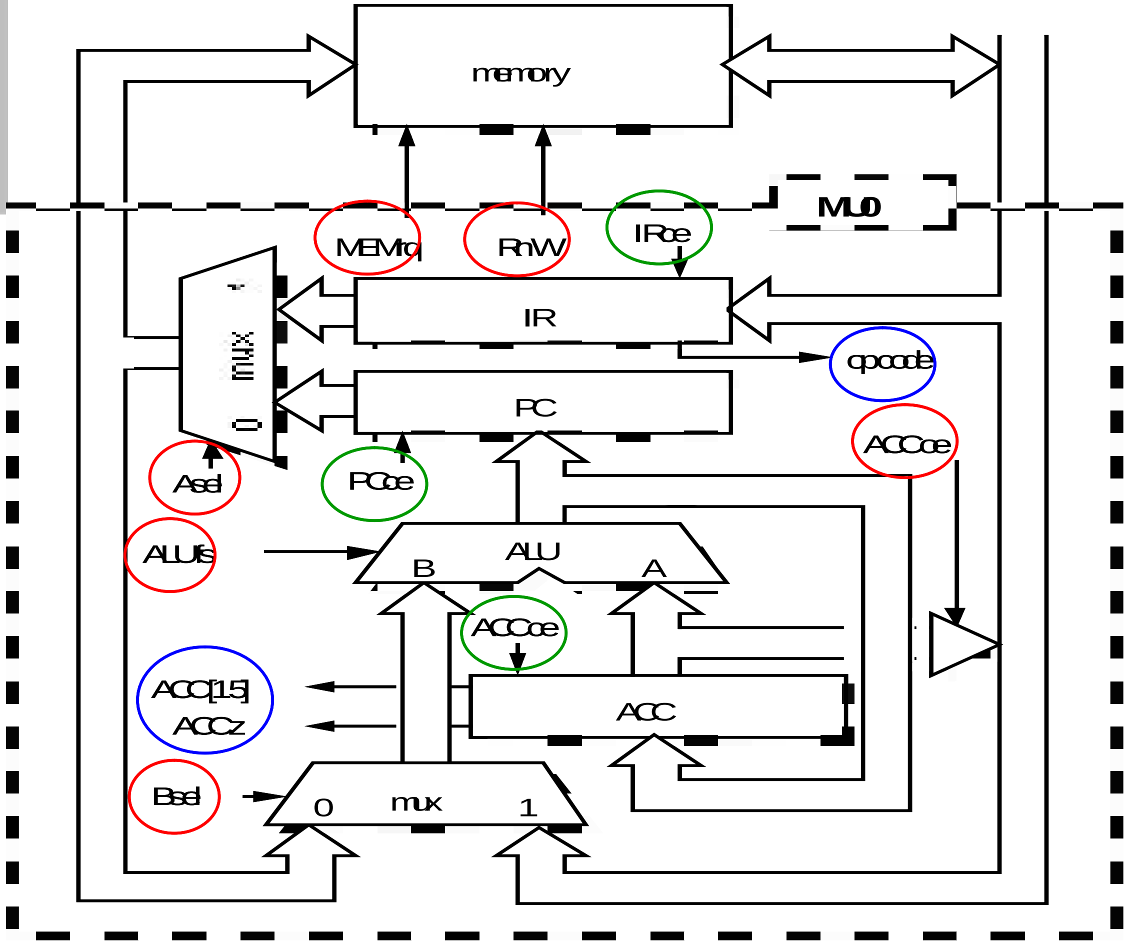
- Datapath operation: An instruction executes in two stages, possibly omitting the first of these;
  - Access the memory operand and perform the desired operation
  - Fetch the next instruction to be executed.

- We will design MU0 to start executing from address 000_{16}. There are several ways to achieve this, one of which is to use the reset signal to zero the ALU output and then clock this into the PC register.

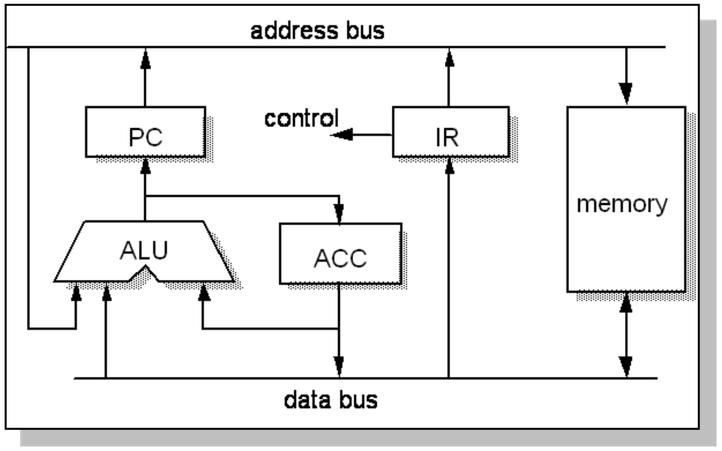
- We assume that all the registers change state on the falling edge of the input clock, and where necessary have control signals to prevent them from changing on a particular clock edge. (PC_{ce}, PC change enable signal)

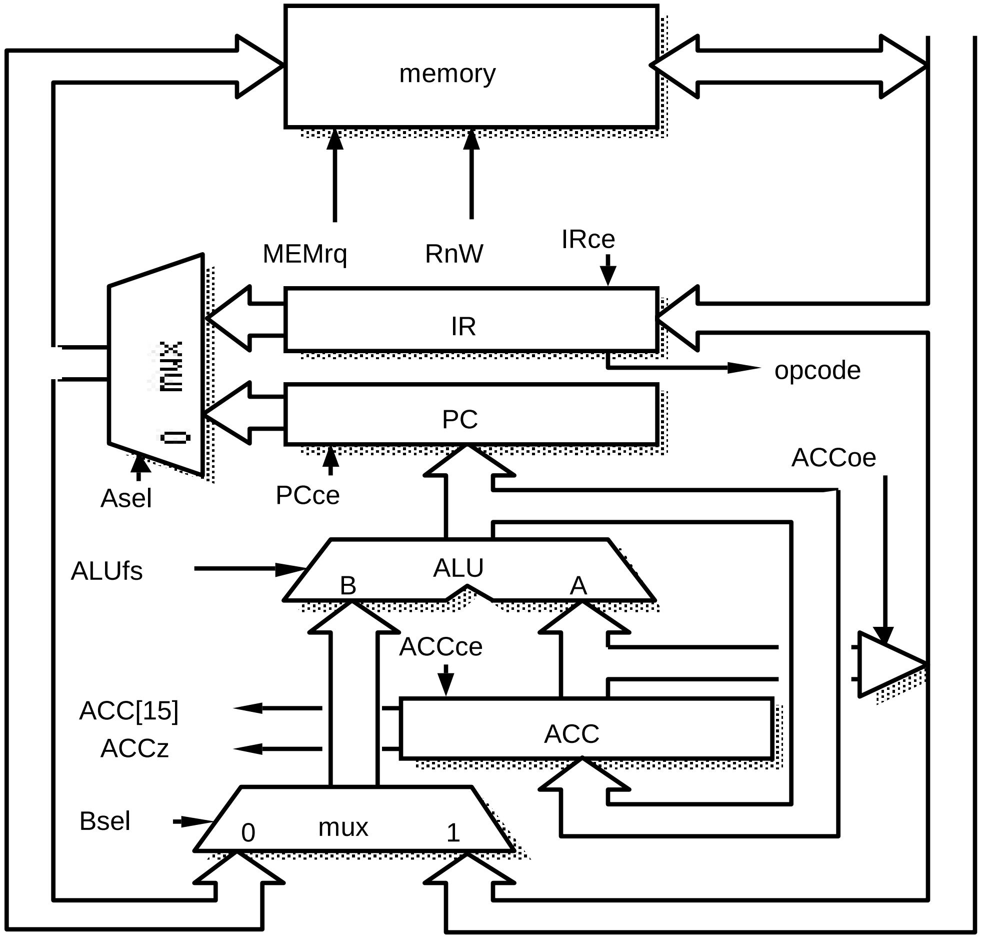
MU0 register transfer level organization





MU0 register transfer level organization





Control Logic

- The control logic simply has to decode the current instruction and generate the appropriate levels on the datapath control signals, using the control inputs from the datapath where necessary.

- The implementation requires only two states, ‘fetch’ and ‘execute’, and one bit of state *(Ex/ft*) is therefore sufficient.

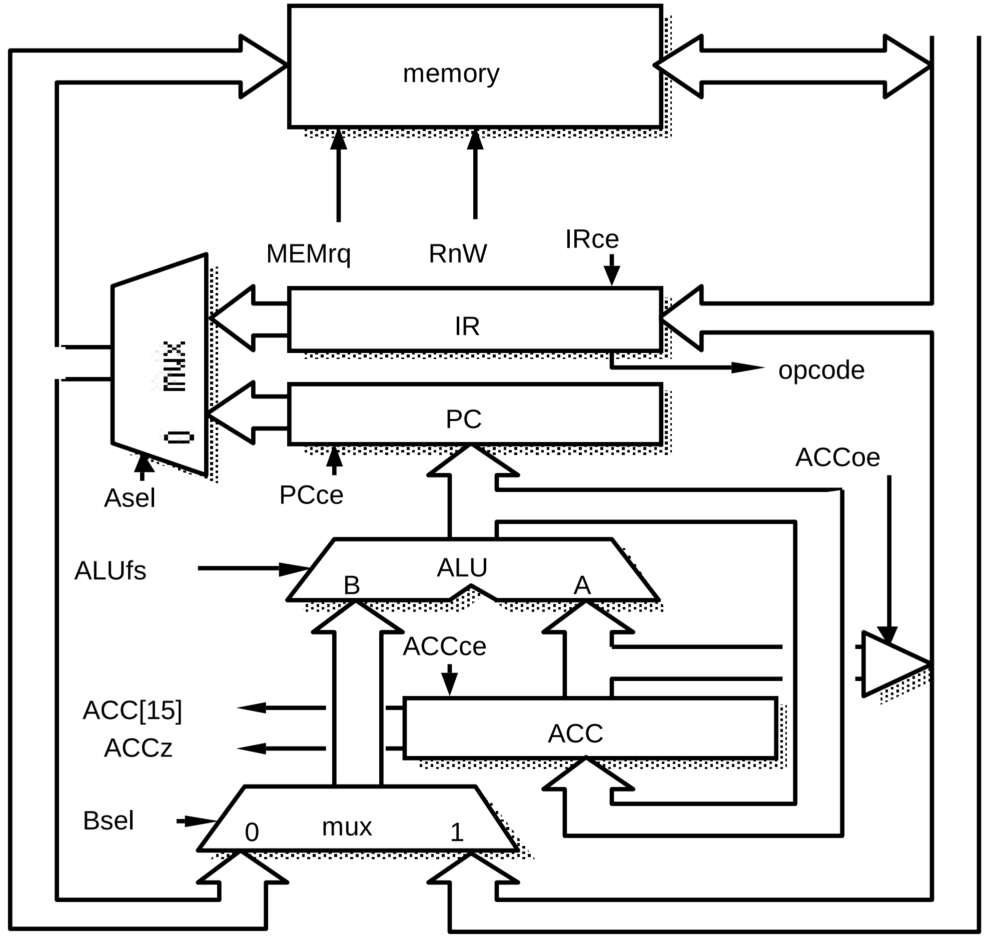
MU0 control logic

- Once the ALU function select codes have been assigned the table may be implemented as a PLA or translated into combinational logic using standard gates.

Inputs						Outputs									
Instruction	Opcode	Reset	Ex/ft	ACCz	ACC15	Asel	Bsel	ACCce	PCce	IRce	ACCoe	ALUfs	MEMrq	RnW	Ex/ft
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

LDA S

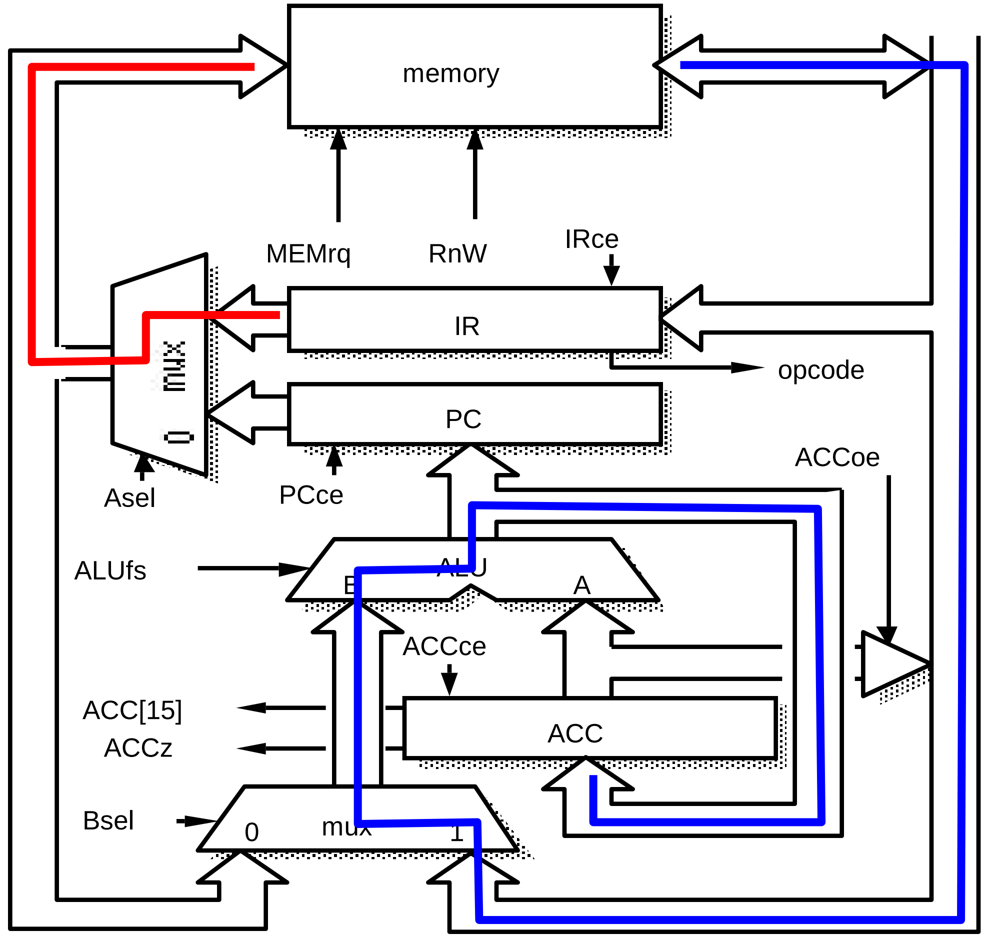
Inputs						Outputs									
Instruction	Opcode	Reset	Ex/ft	ACCz	ACC15	Asel	Bsel	ACCce	PCce	IRce	ACCoe	ALUfs	MEMrq	RnW	Ex/ft
LDA S	0000	0	0	x	x	1	1	1	0	0	0	= B	1	1	1
	0000	0	1	x	x	0	0	0	1	1	0	B+1	1	1	0



100	LDA S
101	ADD S
102	STO S

LDA S (ex)

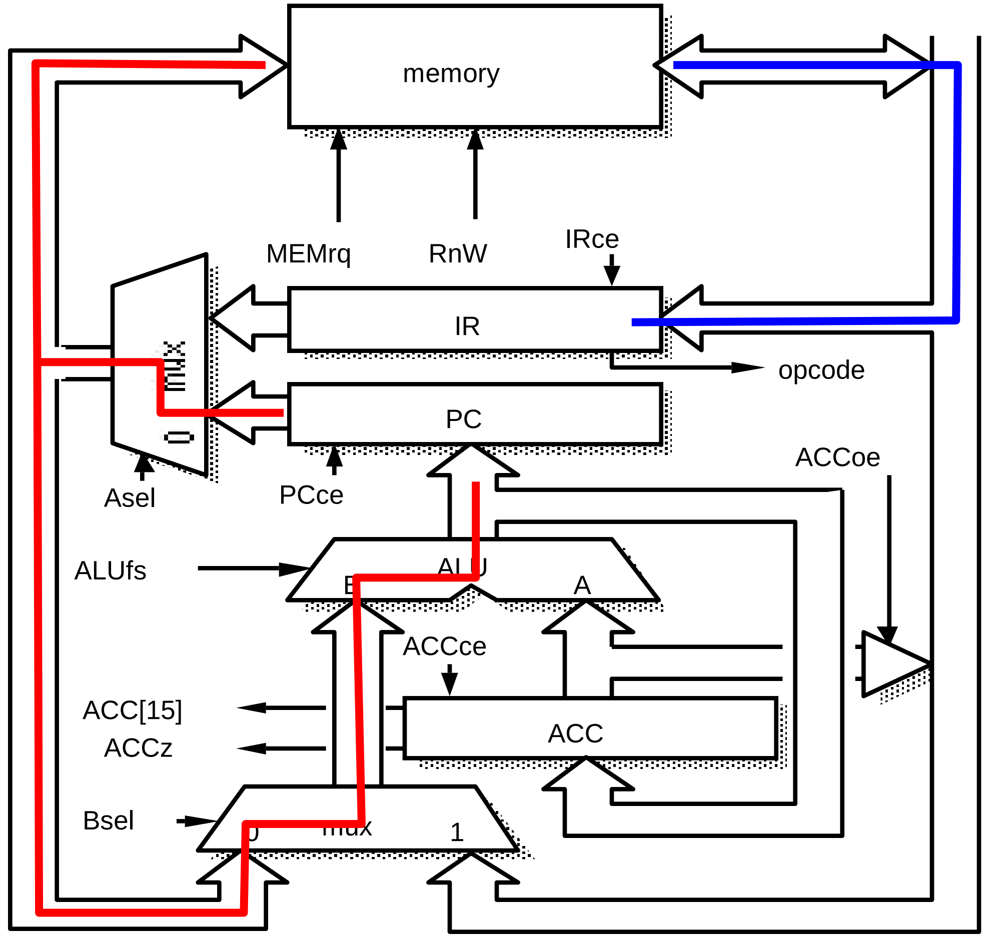
Inputs						Outputs									
Instruction	Opcode	Reset	Ex/ft	ACCz	ACC15	Asel	Bsel	ACCce	PCce	IRce	ACCoe	ALUfs	MEMrq	RnW	Ex/ft
LDA S	0000	0	0	x	x	1	1	1	0	0	0	= B	1	1	1
	0000	0	1	x	x	0	0	0	1	1	0	B+1	1	1	0



100	LDA S
101	ADD S
102	STO S

LDA S (ft)

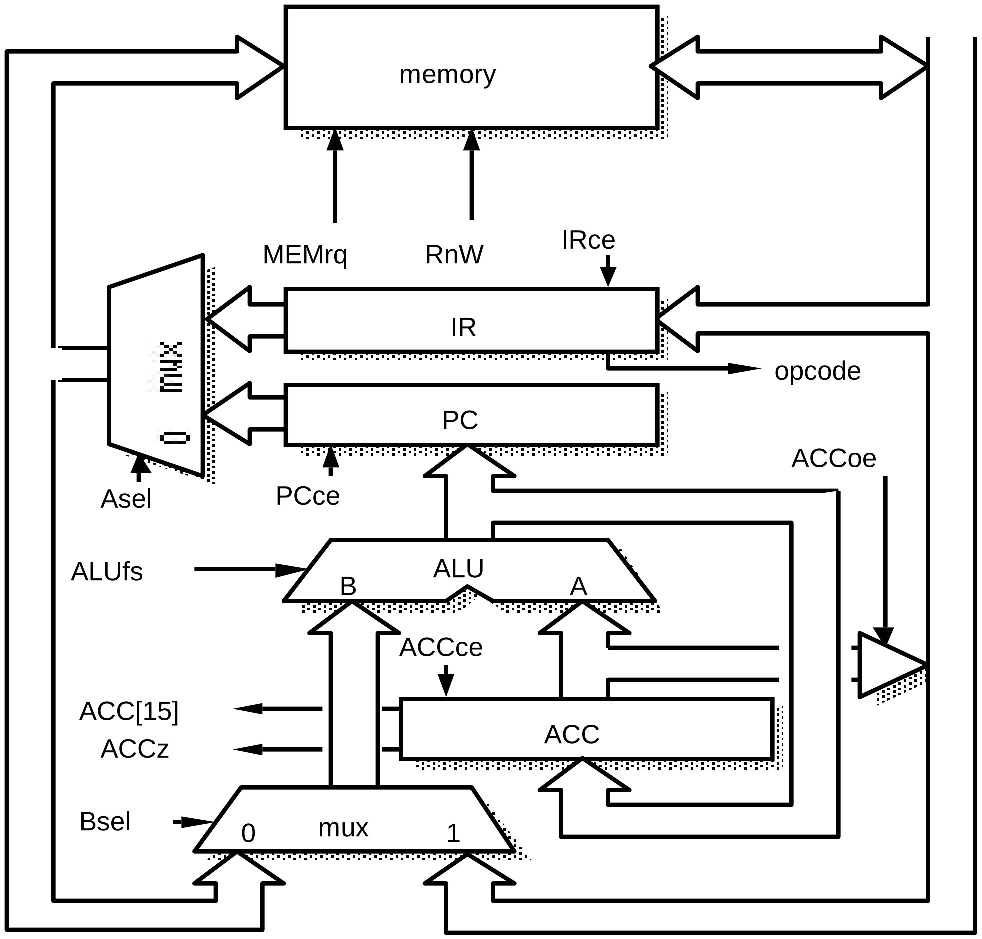
Inputs						Outputs									
Instruction	Opcode	Reset	Ex/ft	ACCz	ACC15	Asel	Bsel	ACCce	PCce	IRce	ACCoe	ALUfs	MEMrq	RnW	Ex/ft
LDA S	0000	0	0	x	x	1	1	1	0	0	0	= B	1	1	1
	0000	0	1	x	x	0	0	0	1	1	0	B+1	1	1	0



100	LDA S
101	ADD S
102	STO S

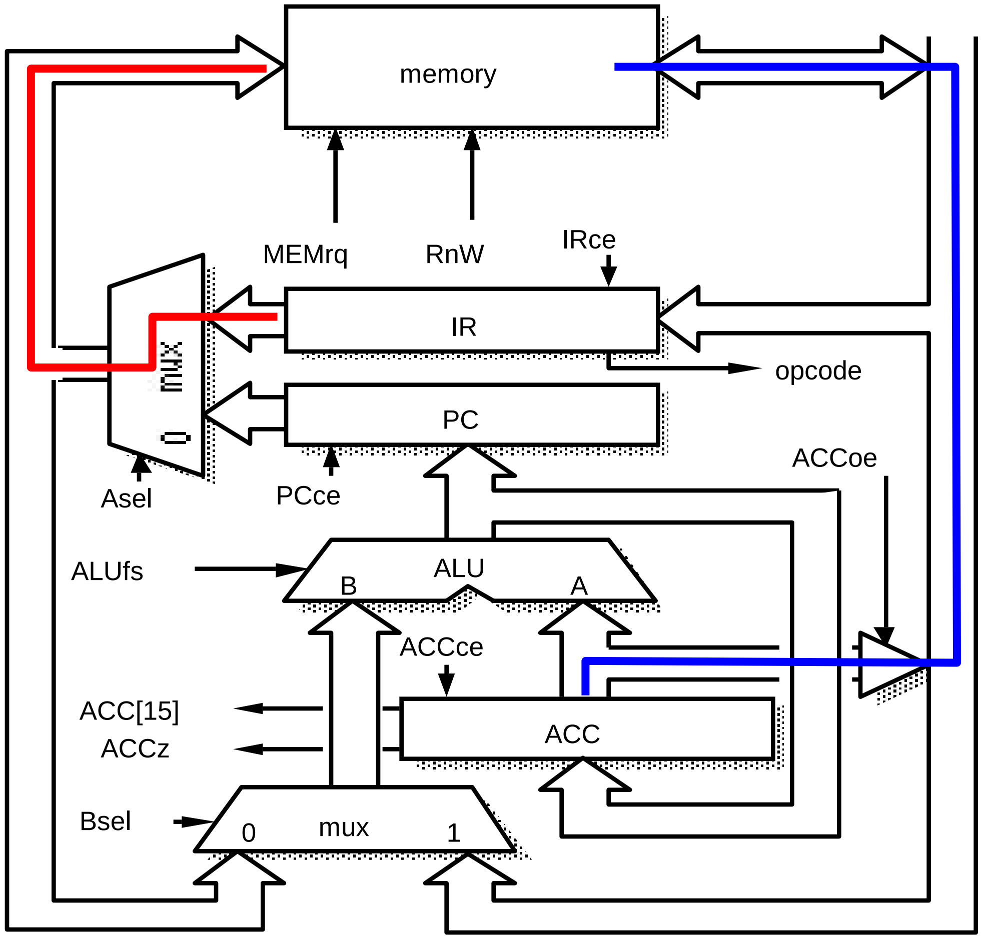
STO S

Instruction	Opcode	Reset	Ex/ft	ACCz	ACC15	Asel	Bsel	ACCce	PCce	IRce	ACCoe	ALUfs	MEMrq	RnW	Ex/ft
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



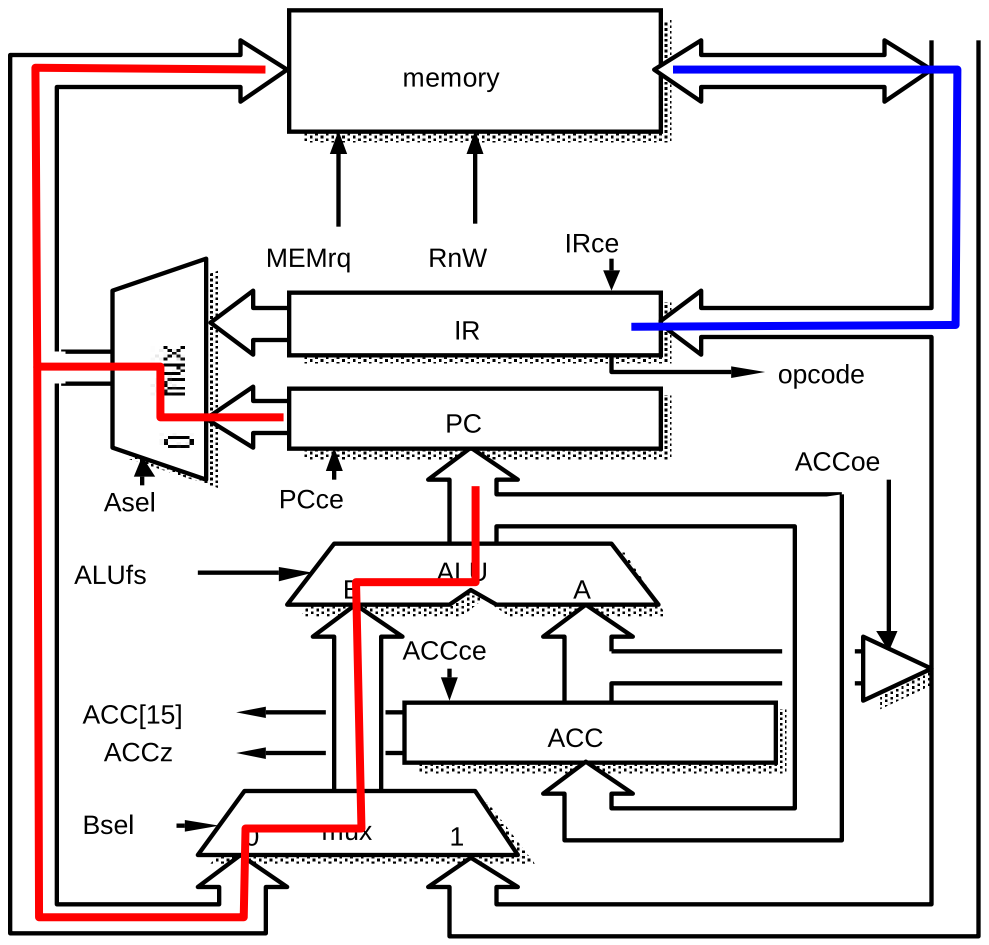
STO S (ex)

Instruction	Opcode	Reset	Ex/ft	ACCz	ACC15	Asel	Bsel	ACCce	PCce	IRce	ACCoe	ALUfs	MEMrq	RnW	Ex/ft
	Inputs					**Outputs**									
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



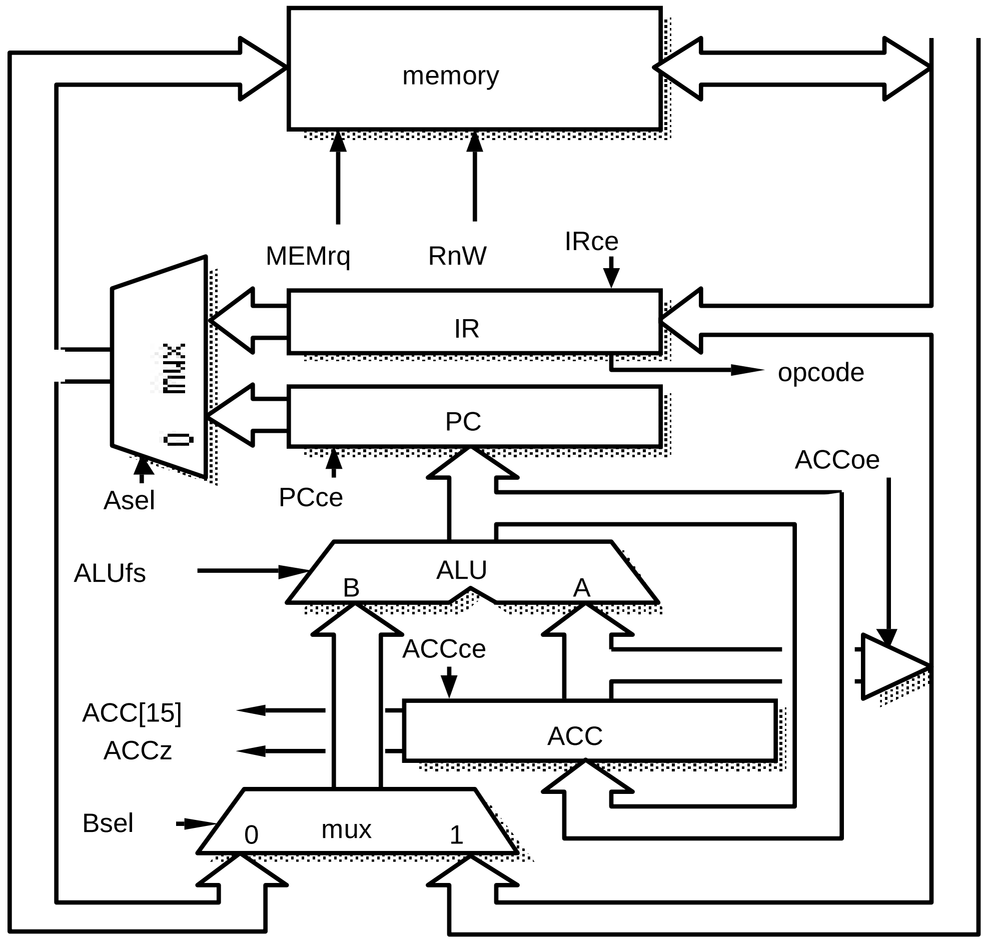
STO S (ft)

Inputs						Outputs									
Instruction	Opcode	Reset	Ex/ft	ACCz	ACC15	Asel	Bsel	ACCce	PCce	IRce	ACCoe	ALUfs	MEMrq	RnW	Ex/ft
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



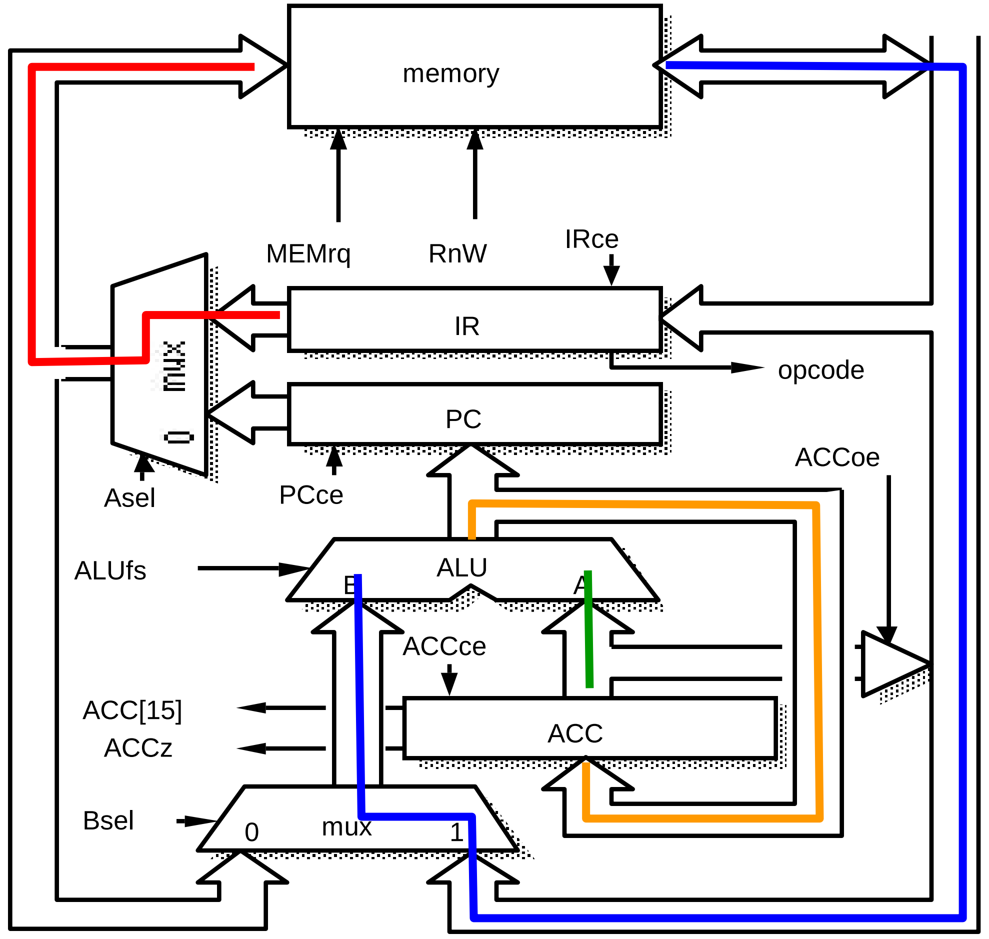
ADD S

Instruction	Opcode	Reset	Ex/ft	ACCz	ACC15	Asel	Bsel	ACCce	PCce	IRce	ACCoe	ALUfs	MEMrq	RnW	Ex/ft
Inputs						Outputs									
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



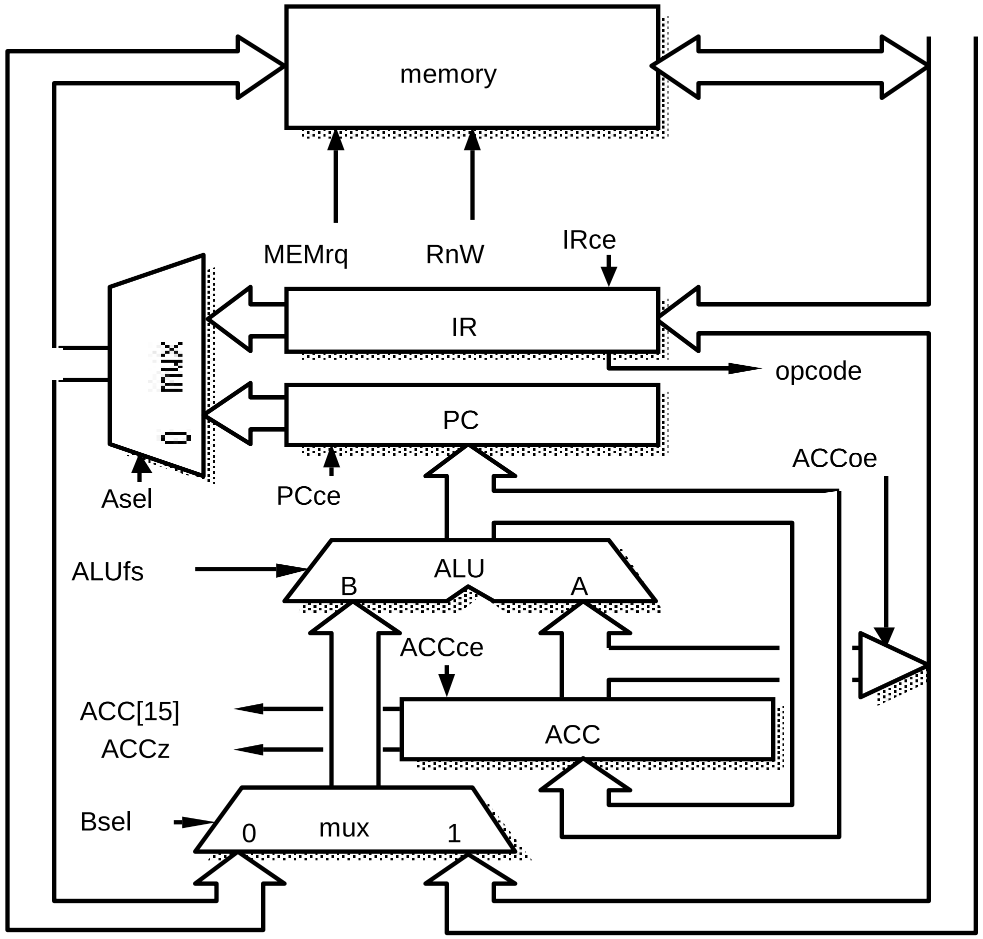
ADD S (ex)

Inputs						Outputs									
Instruction	Opcode	Reset	Ex/ft	ACCz	ACC15	Asel	Bsel	ACCce	PCce	IRce	ACCoe	ALUfs	MEMrq	RnW	Ex/ft
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



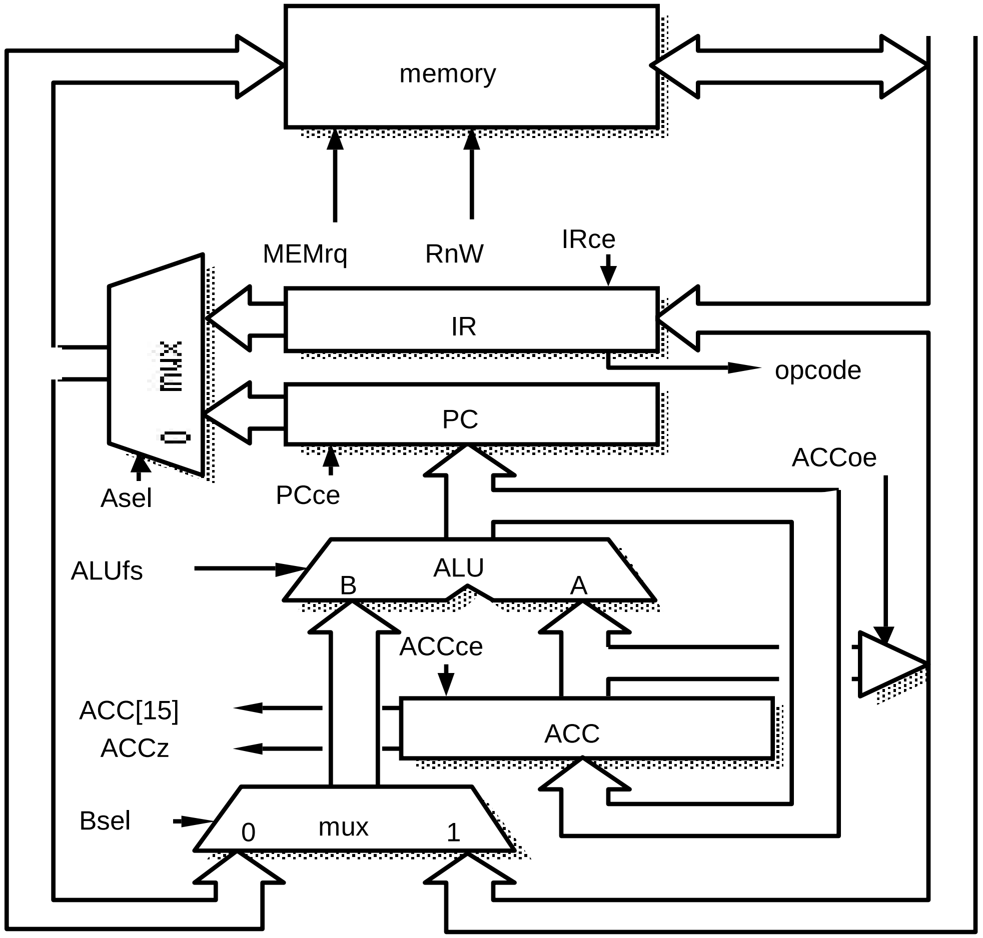
SUB S

Inputs						Outputs									
Instruction	Opcode	Reset	Ex/ft	ACCz	ACC15	Asel	Bsel	ACCce	PCce	IRce	ACCoe	ALUfs	MEMrq	RnW	Ex/ft
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 & JGE & JNE

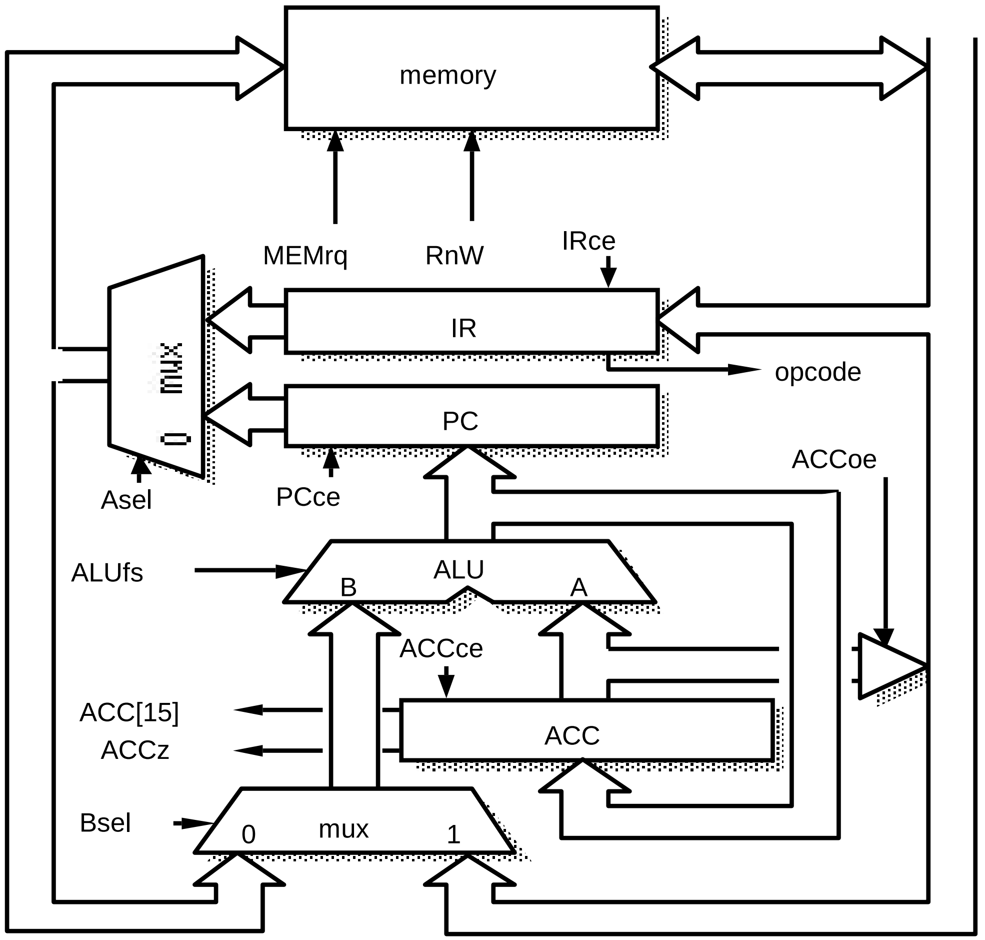
Inputs						Outputs									
Instruction	Opcode	Reset	Ex/ft	ACCz	ACC15	Asel	Bsel	ACCce	PCce	IRce	ACCoe	ALUfs	MEMrq	RnW	Ex/ft
JMP S	0100	0	x	x	x	1	0	0	1	1	0	B+1	1	1	0
JGE S	0101	0	x	x	0	1	0	0	1	1	0	B+1	1	1	0
	0101	0	x	x	1	0	0	0	1	1	0	B+1	1	1	0
JNE S	0110	0	x	0	x	1	0	0	1	1	0	B+1	1	1	0
	0110	0	x	1	x	0	0	0	1	1	0	B+1	1	1	0



```
100   JMP 200
101   ADD S
        .
        .
200   STO  S
201   SUB  S
```

JMP

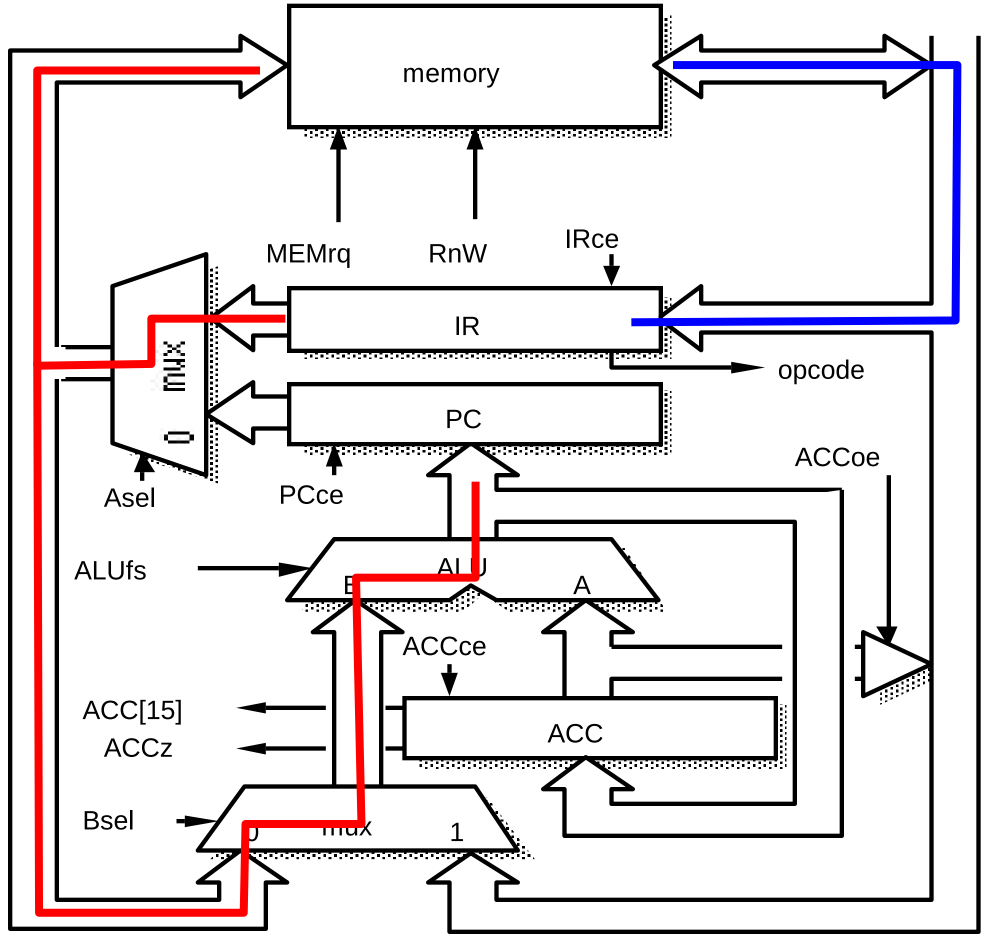
Inputs						Outputs									
Instruction	Opcode	Reset	Ex/ft	ACCz	ACC15	Asel	Bsel	ACCce	PCce	IRce	ACCoe	ALUfs	MEMrq	RnW	Ex/ft
JMP S	0100	0	x	x	x	1	0	0	1	1	0	B+1	1	1	0



```
100   JMP 200
101   ADD S
        .
        .
200   STO  S
201   SUB  S
```

JMP

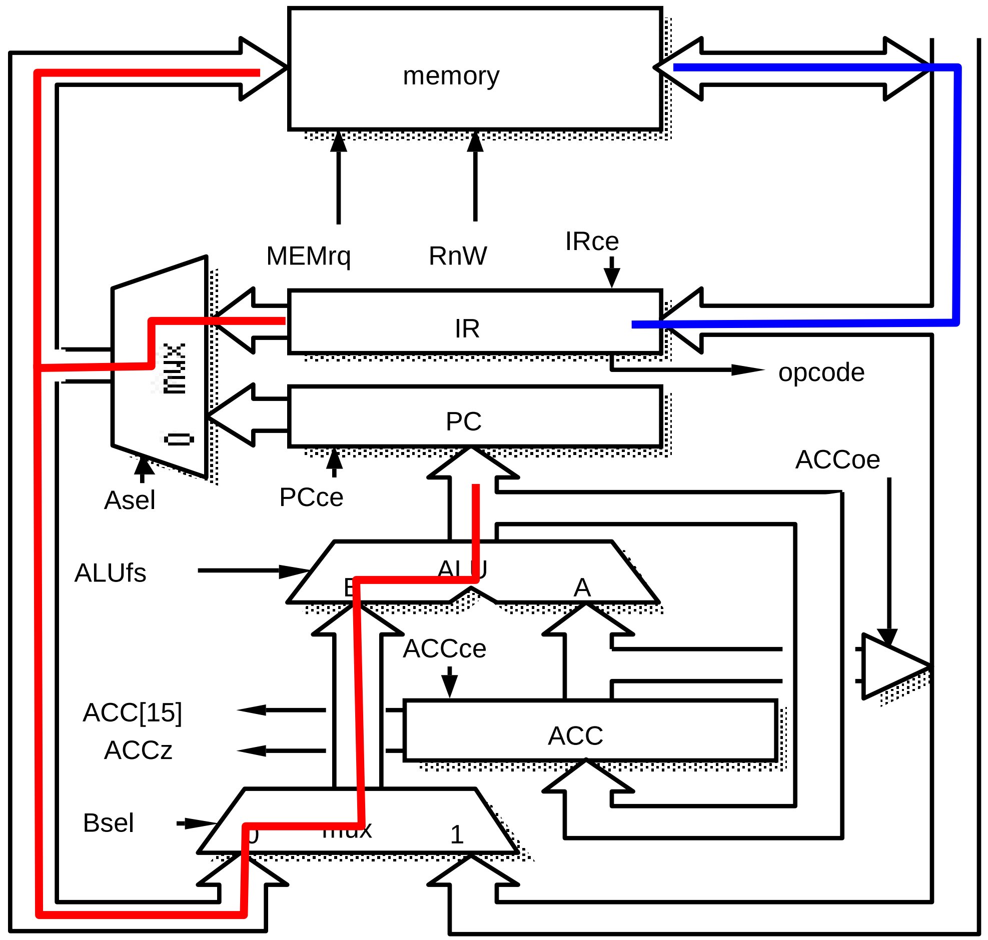
Inputs						Outputs									
Instruction	Opcode	Reset	Ex/ft	ACCz	ACC15	Asel	Bsel	ACCce	PCce	IRce	ACCoe	ALUfs	MEMrq	RnW	Ex/ft
JMP S	0100	0	x	x	x	1	0	0	1	1	0	B+1	1	1	0



100 **JMP 200**
101 **ADD S**
.
.
200 **STO S**
201 **SUB S**

JGE & JNE (taken)

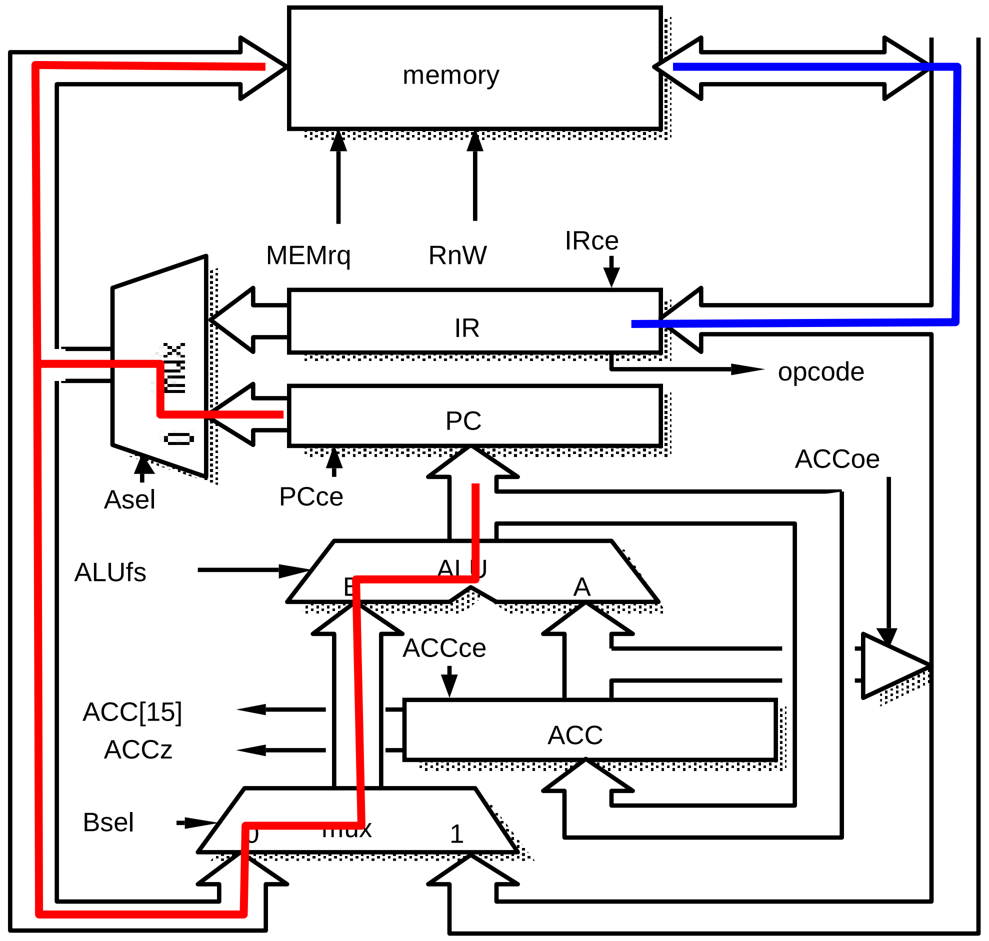
	Inputs					Outputs									
Instruction	Opcode	Reset	Ex/ft	ACCz	ACC15	Asel	Bsel	ACCce	PCce	IRce	ACCoe	ALUfs	MEMrq	RnW	Ex/ft
JGE S	0101	0	x	x	0	1	0	0	1	1	0	B+1	1	1	0
	0101	0	x	x	1	0	0	0	1	1	0	B+1	1	1	0
JNE S	0110	0	x	0	x	1	0	0	1	1	0	B+1	1	1	0
	0110	0	x	1	x	0	0	0	1	1	0	B+1	1	1	0



100 JGE 200
101 ADD S
.
.
200 STO S
201 SUB S

JGE & JNE (not taken)

Inputs						Outputs									
Instruction	Opcode	Reset	Ex/ft	ACCz	ACC15	Asel	Bsel	ACCce	PCce	IRce	ACCoe	ALUfs	MEMrq	RnW	Ex/ft
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



100 JGE 200
101 ADD S
.
.
200 STO S
201 SUB S

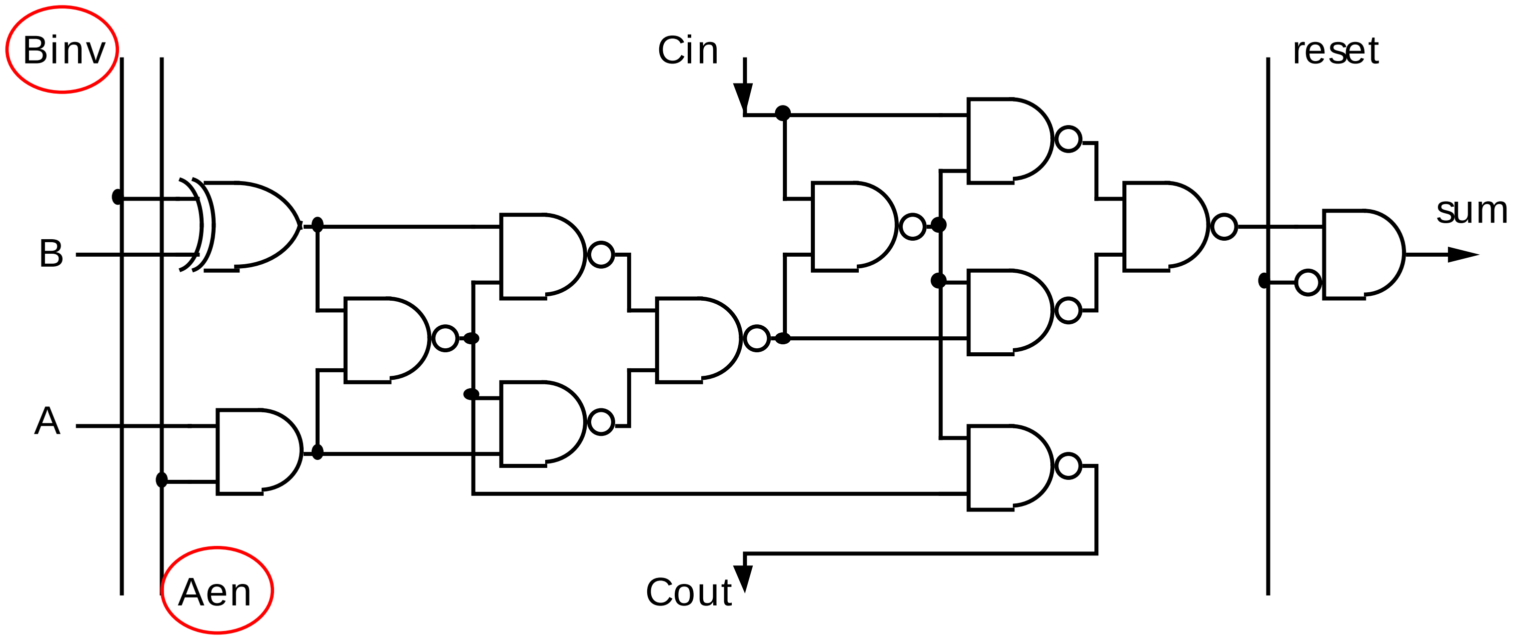
ALU Design

- The MU0 ALU is a little more complex than the simple adder.
- From Table 1.2, there are five ALU functions, *(A+B, A-B, B, B+1, 0)*, the last of which is only used while reset is active. Therefore the reset signal can control this function directly and the control logic need only generate a 2-bit function select code to choose between the other four.

- A+B: normal adder (carry-in is zero)
- A-B: implemented as A+B'+1, requiring invert B input and force carry-in to one.
- B: forcing the A input and the carry-in to zero
- B+1: forcing A to zero and the carry-in to one.

MU0 ALU logic for one bit

- *Aen* enables the A operand or force it to zero
- *Binv* controls whether or not the B operand is inverted.
- Function *(A+B, A-B, B, B+1, 0)*,



MU0 extensions

- MU0 is a very simple processor and would not make a good target for a high-level language compiler, it serves to illustrate the basic principles of processor design. The design process used to develop the first ARM processors differed mainly in complexity and not in principle.

- To turn MU0 into a useful processor takes a lot of work. The following extensions seem most important:

  - Extending the address space.
  - Adding more addressing modes.
  - Allowing the PC to be saved to support a subroutine mechanism
  - Adding more registers, supporting interrupts, and so on…