Processor Design

Georges-Axel Jaloyan

Information Security Group, École normale supérieure

1 December 2020

What you know

- Binary, hexadecimal, octal (read, spoken, written).
- Boolean algebra (read, spoken, written).
- Basic gates, net-lists, basic circuit design (adders, multipliers, ...).
- How to program a netlist simulator.

What we'll see today

- Reminder on how to build an ALU.
- How to insert this circuit in a even bigger one that does everything (note: this is called a processor)
- How to make the processor configurable so that it runs several programs (hello assembly and memory).
- How to make the processor interactive with inputs and outputs.
- What kind of horrendous things we may expect in real life.

Comprehensive bibliography

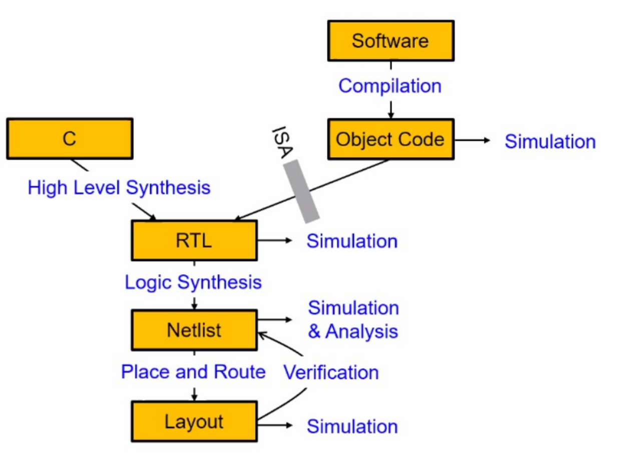
Patterson and Hennessy,
Computer organization and design,
chapters 2 and 4.

Comprehensive bibliography

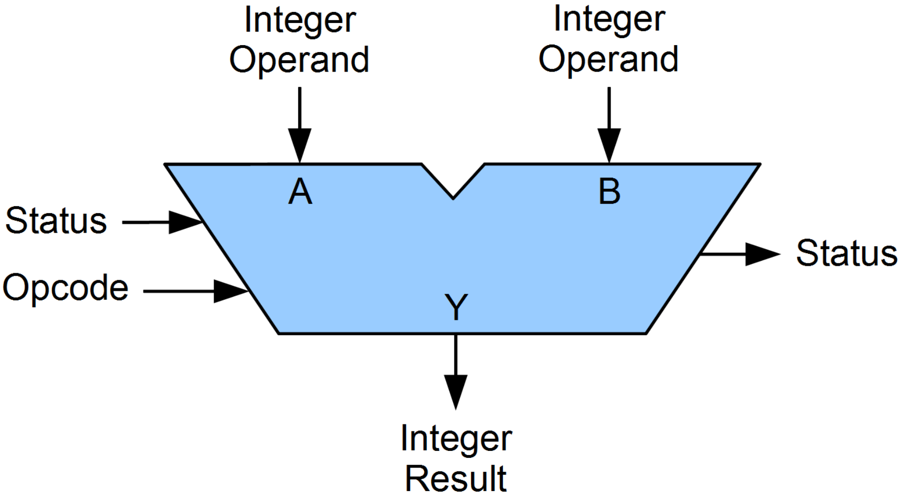
Patterson and Hennessy,
Computer organization and design,
chapters 2 and 4.

That's it.

Software vs hardware



The ALU


Integer
Operand
Integer
Operand
A
B
Status
Opcode
Status
Y
Integer
Result

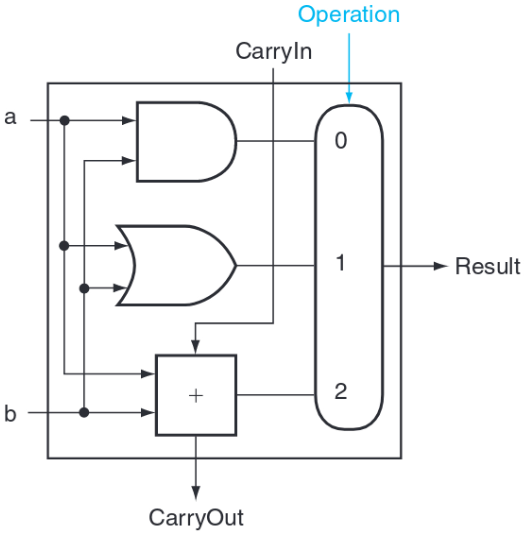
First naïve idea (draw here)

Problems with the naïve idea

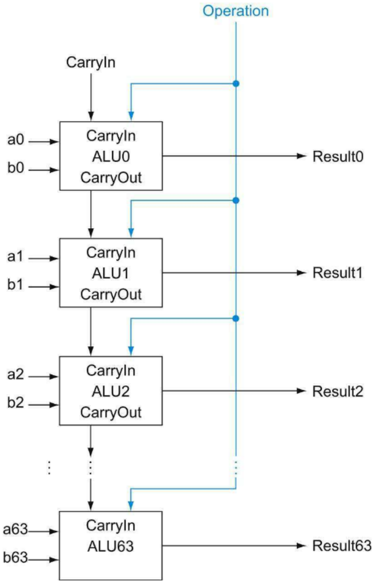
- Too big!
- Too slow!
- Too bad!

In practice, many operations are variants of each other!

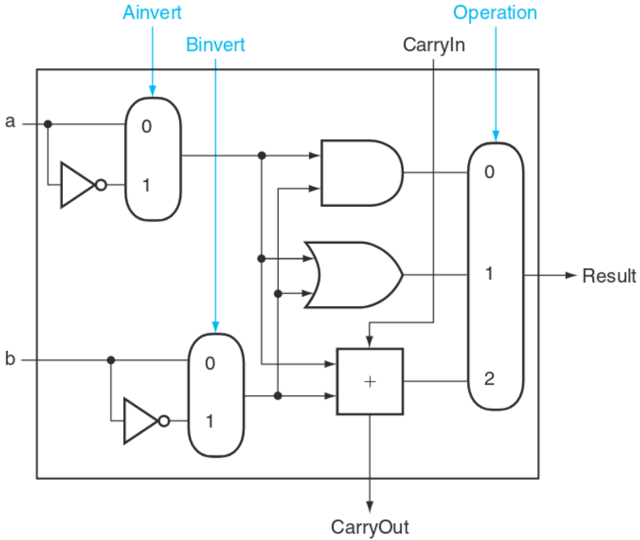
ALU 101: 1-bit



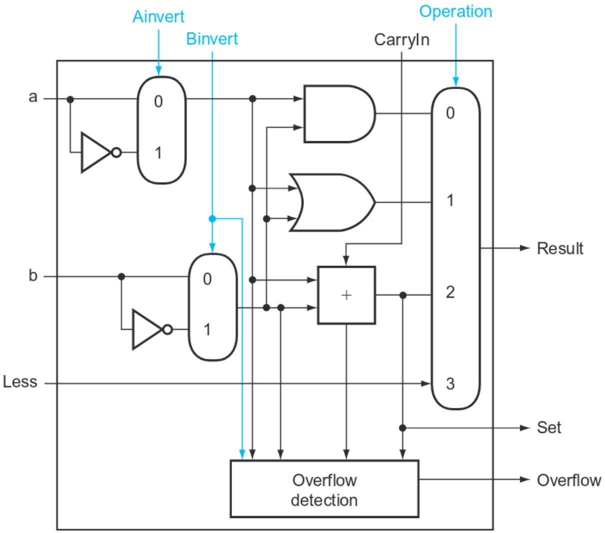
ALU 101: n-bit


Operation
CarryIn
a0
b0
CarryIn
ALU0
CarryOut
Result0
a1
b1
CarryIn
ALU1
CarryOut
Result1
a2
b2
CarryIn
ALU2
CarryOut
Result2
a63
b63
CarryIn
ALU63
Result63

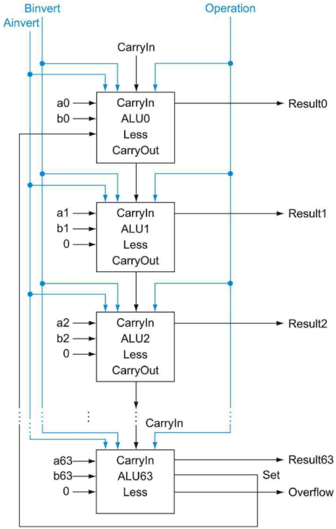
ALU 101: let's add more features



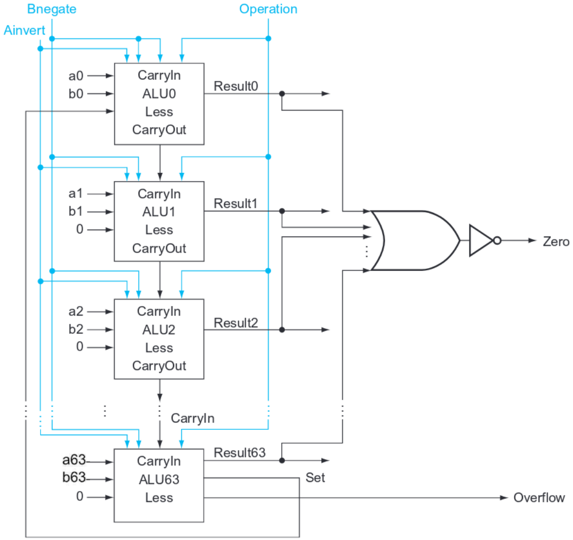
ALU 101: let's add more features



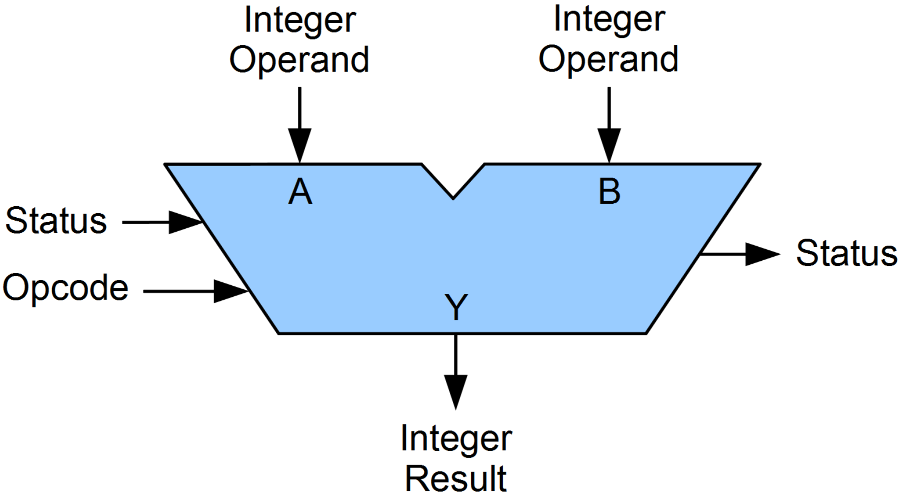
ALU 101: let's add more features



Final ALU


Bnegate
Ainvert
Operation
a0
b0
CarryIn
ALU0
Less
CarryOut
Result0
a1
b1
0
CarryIn
ALU1
Less
CarryOut
Result1
a2
b2
0
CarryIn
ALU2
Less
CarryOut
Result2
CarryIn
a63
b63
0
CarryIn
ALU63
Less
Result63
Set
Overflow
Zero

Final ALU


Integer
Operand
Integer
Operand
A
B
Status
Opcode
Status
Y
Integer
Result

Assembly 101

Two ways for storing data in a computer:

- Registers: their width is 1 word, and are located in the processor. Very fast access (1 cycle), but limited in number. Can be accessed through their name.
- Volatile memory: much bigger, but slower. Can be access through their addresses (pointers in C). Addresses are 1 word wide, and thus can be stored in registers.
- Others: ROM, peripherals, ... Often accessed with addresses. Strange things may happen sometimes.

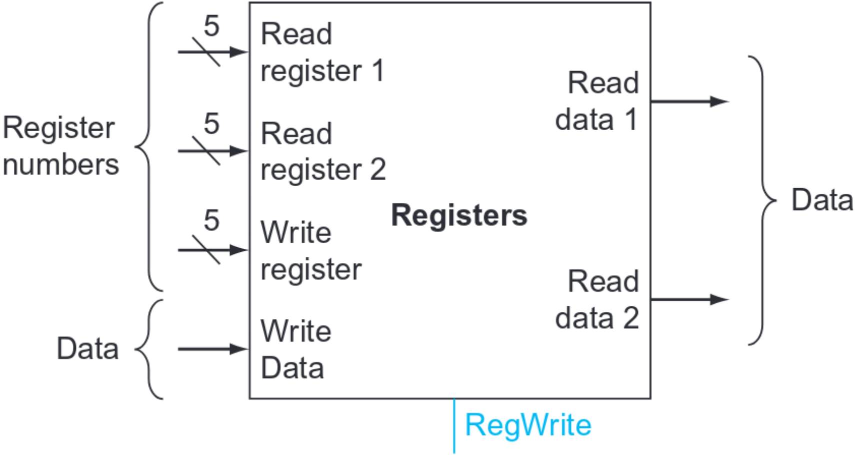
More on registers

Each *instruction set architecture* (ISA) defines the number of registers, their width, and their name:

- x86:32: 4 general purpose 32 bits registers (eax, ebx, ecx, edx), 6 segment registers (cs, ds, es, fs, gs, ss)
- rv64gc: 31 general purpose 64 bits registers (x1-x31) + 2 special registers (x0 et pc) + ...
- aarch64: 31 general purpose 64 bits registers (x0-x30) + 4 special registers (zr, sp, pc, elr) + ...

How to implement (assembly) 1-bit registers (draw)

How to implement (assembly) registers



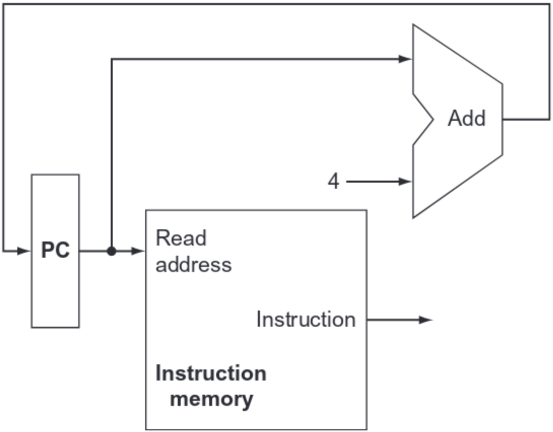
To sum it up

- We know how to make an ALU
- We know how to get the value of registers from their register number.

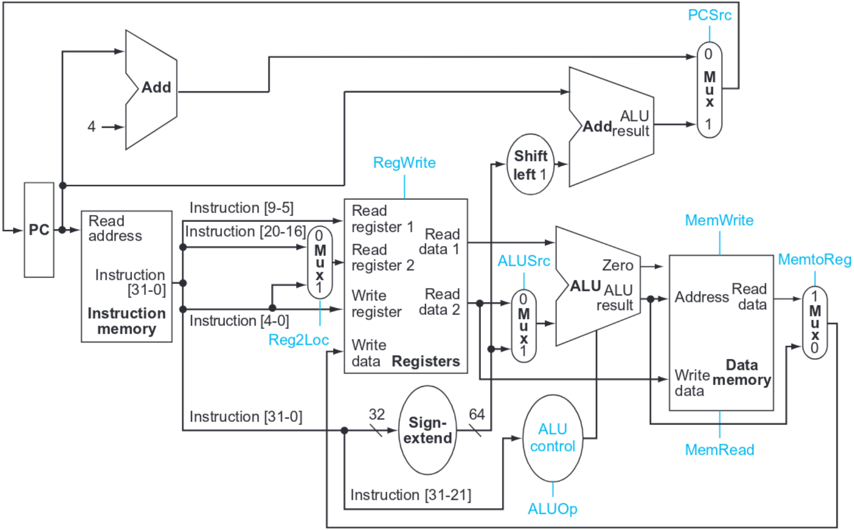
We now need to:

- Get an instruction from memory.
- Find what type of instruction it is.
- Read and write to/from memory.
- Handle jumps.

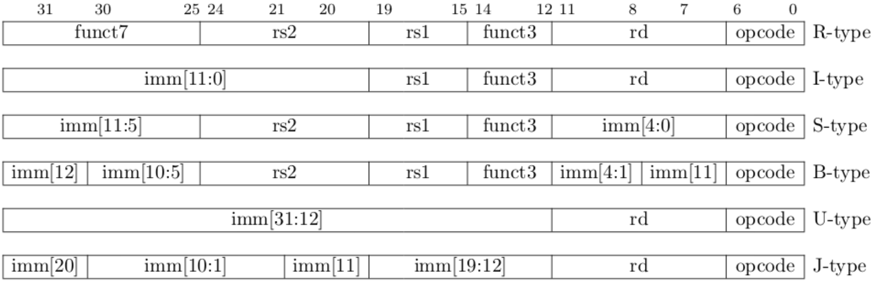
Where are my instructions?



Piping this into the ALU + registers

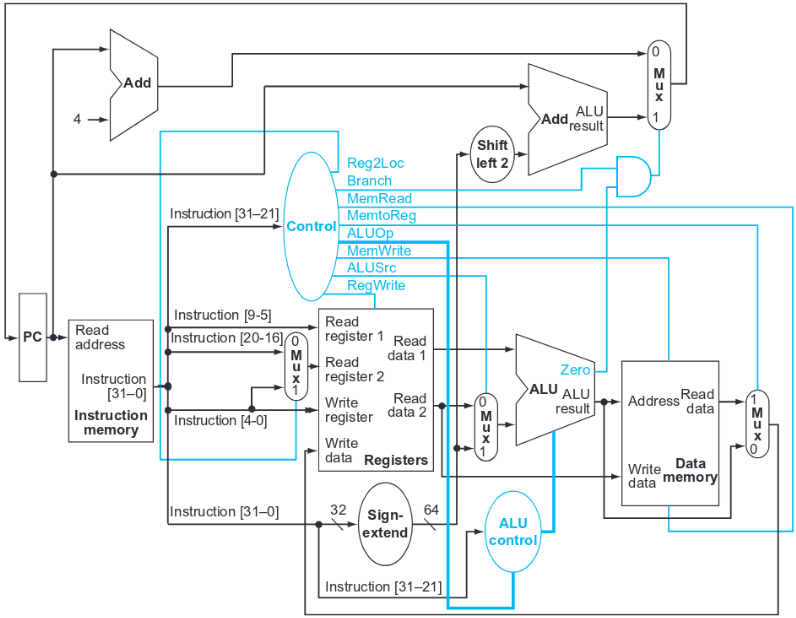


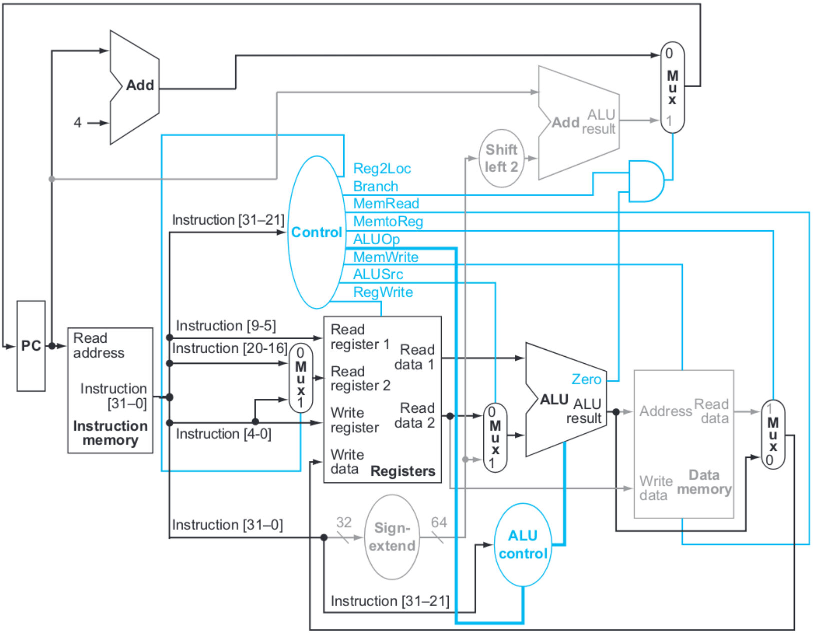
What about instructions

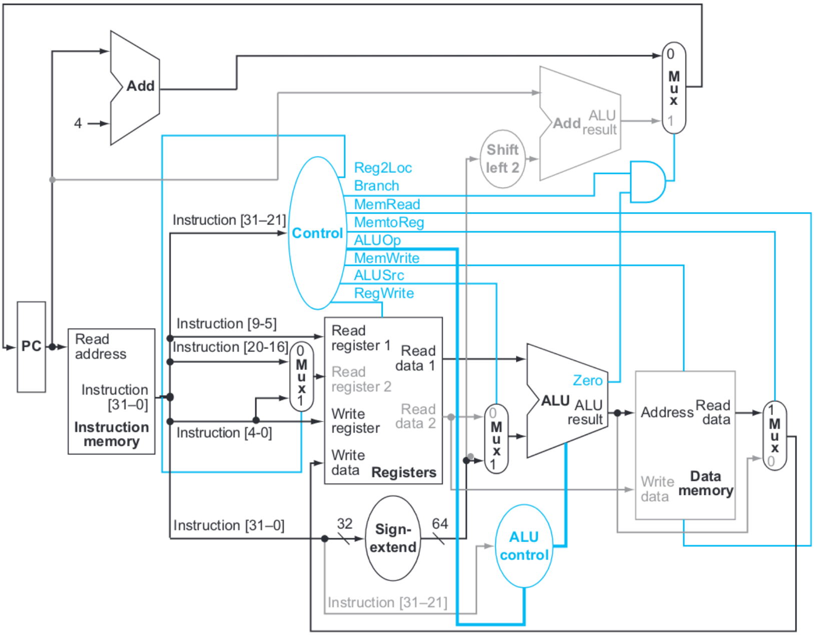


31	30 25	24 21	20	19 15	14 12	11 8	7	6 0	
funct7		rs2		rs1	funct3	rd		opcode	R-type
imm[11:0]				rs1	funct3	rd		opcode	I-type
imm[11:5]		rs2		rs1	funct3	imm[4:0]		opcode	S-type
imm[12]	imm[10:5]	rs2		rs1	funct3	imm[4:1]	imm[11]	opcode	B-type
imm[31:12]						rd		opcode	U-type
imm[20]	imm[10:1]		imm[11]	imm[19:12]		rd		opcode	J-type

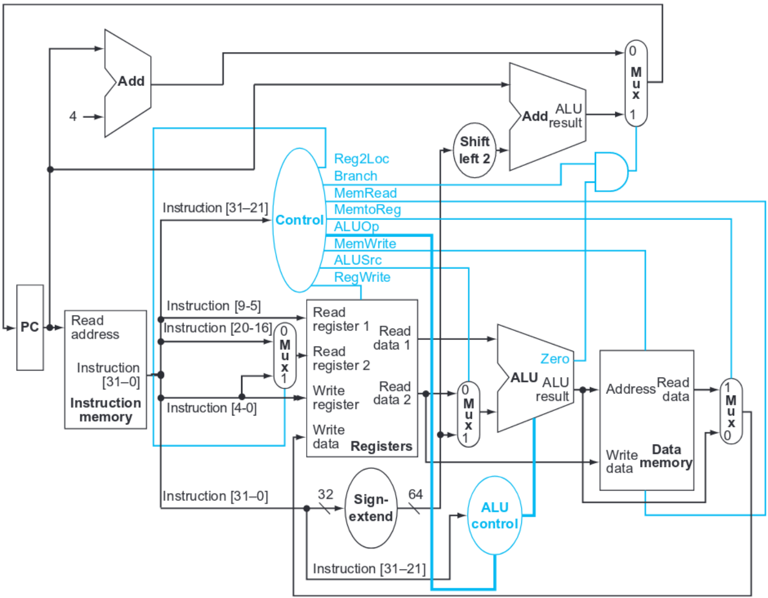
Case study: RISC-V reference manual


Add
4
Add
ALU
result
Shift
left 2
0
M
u
x
1
Reg2Loc
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [31–21]
Control
PC
Read
address
Instruction
[31–0]
Instruction
memory
Instruction [9-5]
Instruction [20-16]
0
M
u
x
1
Instruction [4-0]
Read
register 1
Read
register 2
Write
register
Write
data
Read
data 1
Read
data 2
Registers
0
M
u
x
1
Zero
ALU
ALU
result
Address
Read
data
Write
data
Data
memory
1
M
u
x
0
Instruction [31–0]
32
Sign-
extend
64
ALU
control
Instruction [31–21]

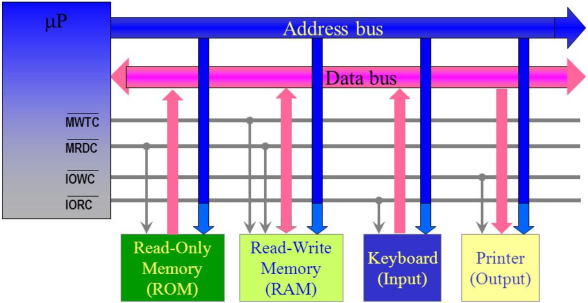

Add
4
0
M
u
x
1
Add
ALU
result
Shift
left 2
Reg2Loc
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31–21]
Instruction [9-5]
Instruction [20-16]
0
M
u
x
1
Instruction [4-0]
PC
Read
address
Instruction
[31–0]
Instruction
memory
Read
register 1
Read
register 2
Write
register
Write
data
Read
data 1
Read
data 2
Registers
0
M
u
x
1
Zero
ALU
ALU
result
Address
Read
data
Write
data
Data
memory
1
M
u
x
0
Instruction [31–0]
32
Sign-
extend
64
ALU
control
Instruction [31–21]

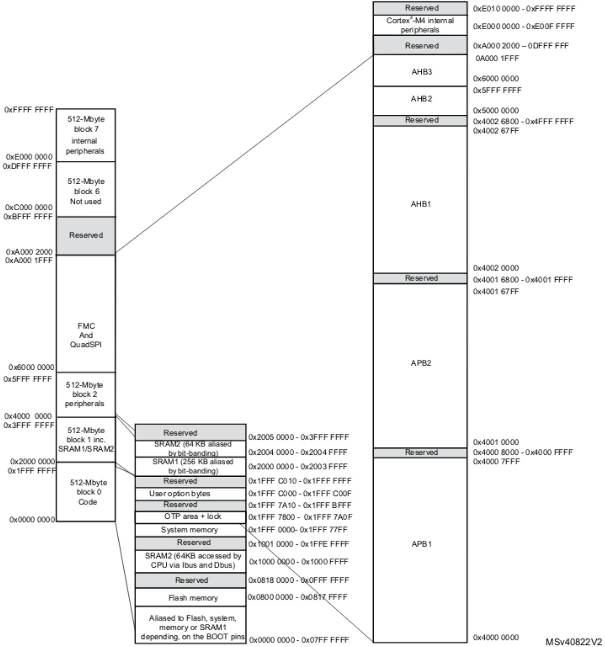

Add
4
Add
ALU
result
Shift
left 2
0
M
u
x
1
Reg2Loc
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31–21]
Instruction [9-5]
Instruction [20-16]
Instruction [4-0]
Instruction [31–0]
PC
Read
address
Instruction
[31–0]
Instruction
memory
Read
register 1
Read
register 2
Write
register
Write
data
Read
data 1
Read
data 2
Registers
Zero
ALU
ALU
result
Address
Read
data
Write
data
Data
memory
32
Sign-
extend
64
ALU
control
Instruction [31–21]

Handling several memory mapped devices



Handling several memory mapped devices





0xFFFF FFFF
512-Mbyte block 7 internal peripherals
0xE000 0000
0xDFFF FFFF
512-Mbyte block 6 Not used
0xC000 0000
0xBFFF FFFF
Reserved
0xA000 2000
0xA000 1FFF
FMC And QuadSPI
0x6000 0000
0x5FFF FFFF
512-Mbyte block 2 peripherals
0x4000 0000
0x3FFF FFFF
512-Mbyte block 1 inc. SRAM1/SRAM2
0x2000 0000
0x1FFF FFFF
512-Mbyte block 0 Code
0x0000 0000

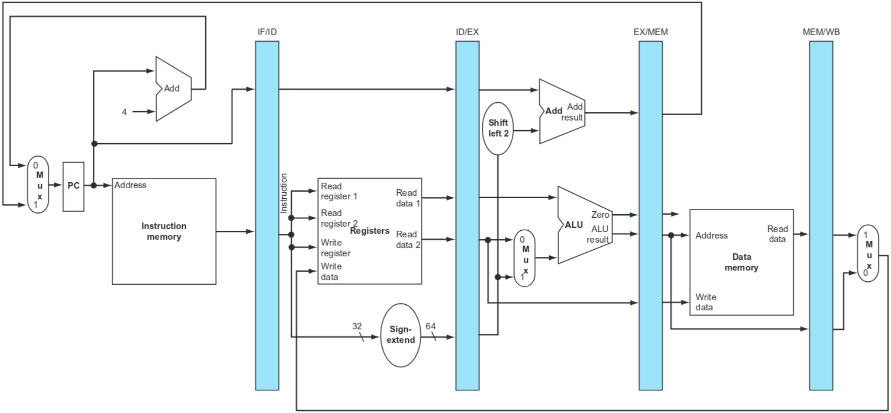
Reserved	0x2005 0000 - 0x3FFF FFFF
SRAM2 (64 KB aliased by bit-banding)	0x2004 0000 - 0x2004 FFFF
SRAM1 (256 KB aliased by bit-banding)	0x2000 0000 - 0x2003 FFFF
Reserved	0x1FFF C010 - 0x1FFF FFFF
User option bytes	0x1FFF C000 - 0x1FFF C00F
Reserved	0x1FFF 7A10 - 0x1FFF BFFF
OTP area + lock	0x1FFF 7800 - 0x1FFF 7A0F
System memory	0x1FFF 0000- 0x1FFF 77FF
Reserved	0x1001 0000 - 0x1FFE FFFF
SRAM2 (64KB accessed by CPU via Ibus and Dbus)	0x1000 0000 - 0x1000 FFFF
Reserved	0x0818 0000 - 0x0FFF FFFF
Flash memory	0x0800 0000 - 0x0817 FFFF
Aliased to Flash, system, memory or SRAM1 depending, on the BOOT pins	0x0000 0000 - 0x07FF FFFF

Reserved 0xE010 0000 - 0xFFFF FFFF
Cortex®-M4 internal peripherals 0xE000 0000 - 0xE00F FFFF
Reserved 0xA000 2000 – 0DFFF FFF
0A000 1FFF
AHB3
0x6000 0000
0x5FFF FFFF
AHB2
0x5000 0000
Reserved 0x4002 6800 - 0x4FFF FFFF
0x4002 67FF
AHB1
0x4002 0000
Reserved 0x4001 6800 - 0x4001 FFFF
0x4001 67FF
APB2
0x4001 0000
Reserved 0x4000 8000 - 0x4000 FFFF
0x4000 7FFF
APB1
0x4000 0000
MSv40822V2

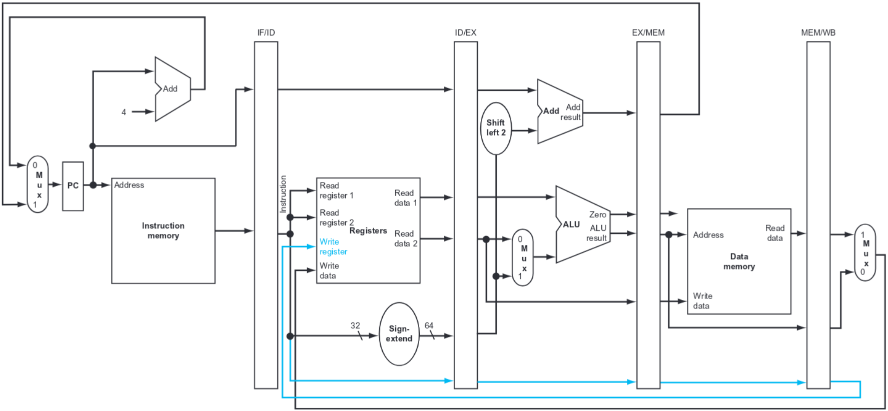
What’s missing

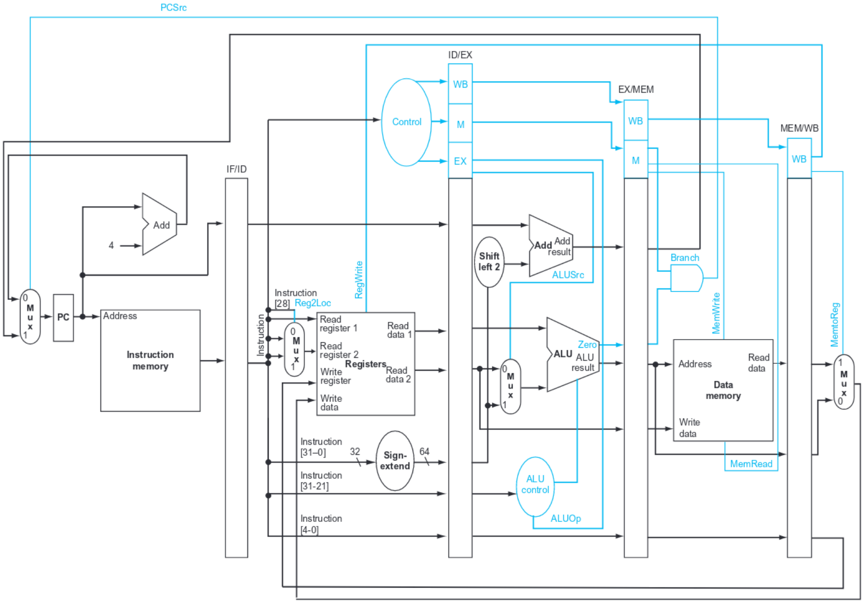
- Multiplication (in the ALU, or in a dedicated ALU)
- Floating point operations (dedicated registers and ALU)
- Little demo on floating points (if we have time)
- Pipelining

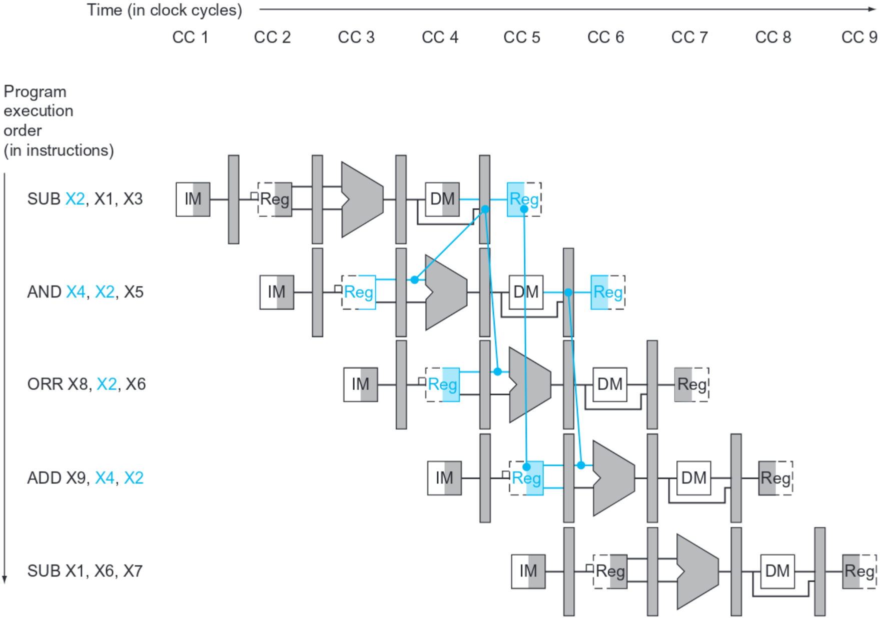
Pipelining


IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
32
Sign-extend
64
Shift left 2
Add
Add result
ALU
Zero
ALU result
0
M
u
x
1
Address
Data memory
Write data
Read data
1
M
u
x
0

Fixing ld


IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
32
Sign-extend
64
Shift left 2
Add
Add result
0
M
u
x
1
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
1
M
u
x
0


PCSrc
ID/EX
EX/MEM
MEM/WB
IF/ID
Control
WB
M
EX
WB
M
WB
Add
4
PC
M u x
0
1
Address
Instruction memory
Instruction
Instruction [28]
Reg2Loc
RegWrite
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
M u x
0
1
Instruction [31–0]
32
Sign-extend
64
Instruction [31-21]
Instruction [4-0]
Shift left 2
Add
Add result
ALUSrc
M u x
0
1
ALU
Zero
ALU result
ALU control
ALUOp
Branch
MemWrite
Address
Read data
Data memory
Write data
MemRead
MemtoReg
M u x
1
0



Time (in clock cycles)

CC 1 CC 2 CC 3 CC 4 CC 5 CC 6 CC 7 CC 8 CC 9

Program execution order (in instructions)

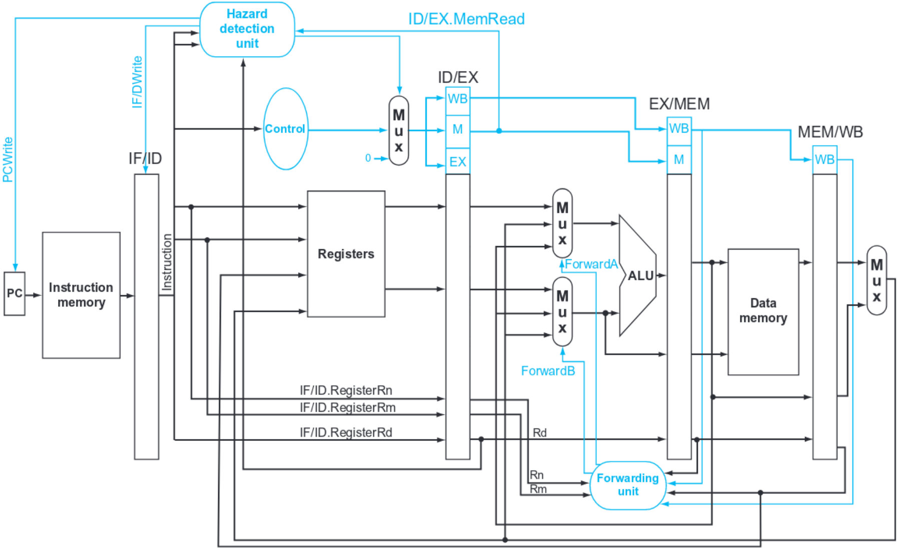
SUB X2, X1, X3

AND X4, X2, X5

ORR X8, X2, X6

ADD X9, X4, X2

SUB X1, X6, X7


Hazard detection unit
ID/EX.MemRead
IF/DWrite
PCWrite
Control
0
M u x
ID/EX
WB
M
EX
EX/MEM
WB
M
MEM/WB
WB
IF/ID
PC
Instruction memory
Instruction
Registers
M u x
ForwardA
ALU
M u x
ForwardB
Data memory
M u x
IF/ID.RegisterRn
IF/ID.RegisterRm
IF/ID.RegisterRd
Rd
Rn
Rm
Forwarding unit

Plus other extras

In practice, it's a little bit more complicated:

- Random instruction set extensions (AVX, SSE, ...)
- Caches
- Exceptions
- Branch prediction
- Virtualized memory
- Privilege levels

Some other processors

https://en.wikichip.org/w/images/1/17/sifive_7_series_block_diagram.svg https://en.wikichip.org/w/images/4/42/tremont_block_diagram.svg

That's all Folks!

Any questions?