EE2016 Microprocessor Theory & Lab
Aug – Nov 2024

Week3: Processor Design – ALU Components

Dr. R. Manivasakan, OSWM lab, IIT, Madras

Recap

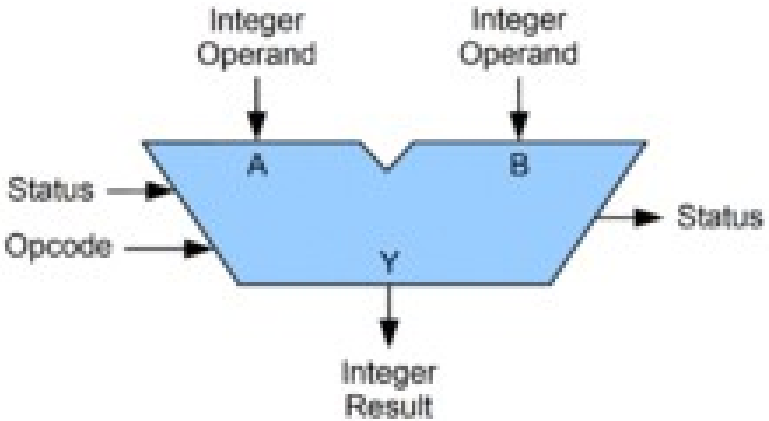
- We have designed the memory from flip-flops and implemented MOV instruction
  - Connected flip-flops through single bit data bus
  - Connected group of flip-flops through multi-bit data bus
  - Register to register data transfer
  - MOV instruction
- We now design the ALU, how it is connected to data bus and how control signals are essential in its operation
- Reference: W. Stallings pages: 15, 18, 303, 305-347, 621-624, 674-676.

Arithmetic & Logic Unit (ALU)

- Does ALL the calculations / computations
  - Arithmetic
    - Handles integers
    - May handle floating point (real) numbers
  - Logical
  - Core function which justifies the word “computer”
- Everything else in the computer is there to service this unit (apart from data flow)
- Advanced Arithmetic functions can be carried out
  - Either by arithmetic routines / algorithms
  - Or a separate FPU (maths co-processor)

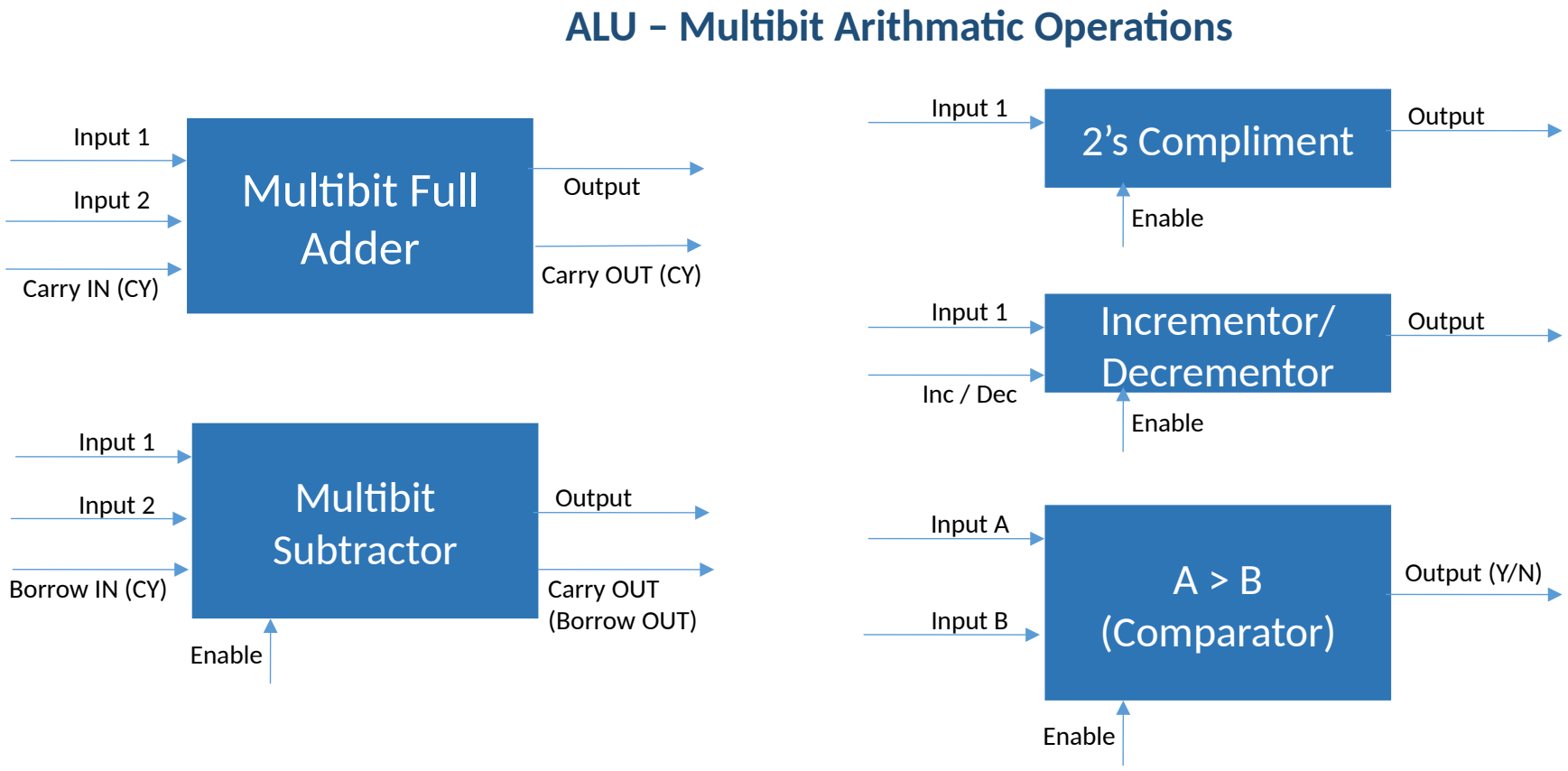
Components of Arithmetic Logic Unit (ALU)

- Arithmetic Computing Engines
  - Architectural (instruction)
    - Eg: ADD, MULT
  - Organizational (Hardware)
    - Eg: Half & full adders, multipliers, Incrementors / decrementors,
- Logical Computing Engines
  - Architectural (instructions)
    - AND, OR, Ex-OR, complements,
  - Organizational (hardware)
    - Eg: Shifter (barrel shifter),



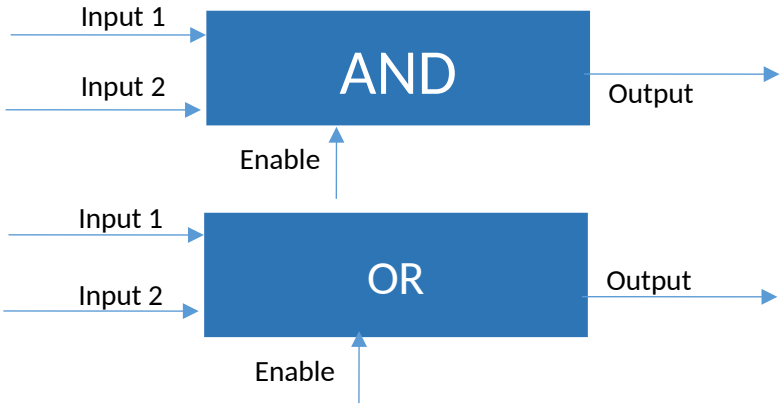
ALU Components

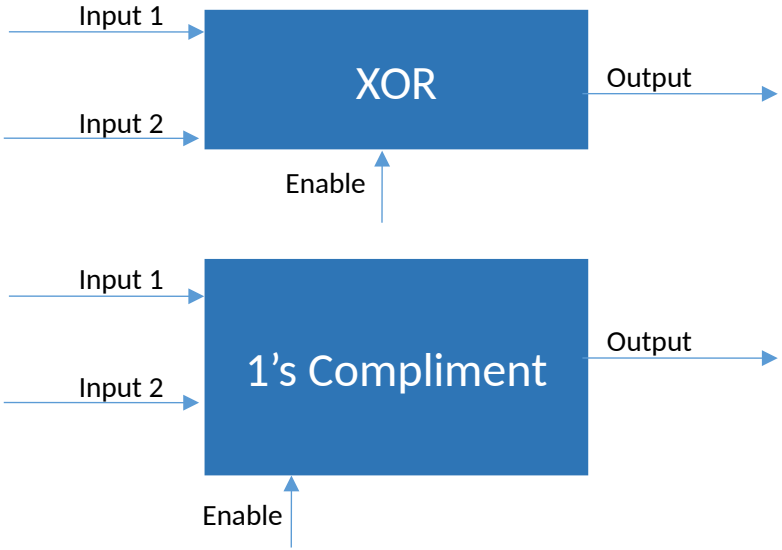
- ALU: Collection of functional units with common inputs and outputs with Output enable (Fn Select)



ALU Components

- Bitwise logical operation

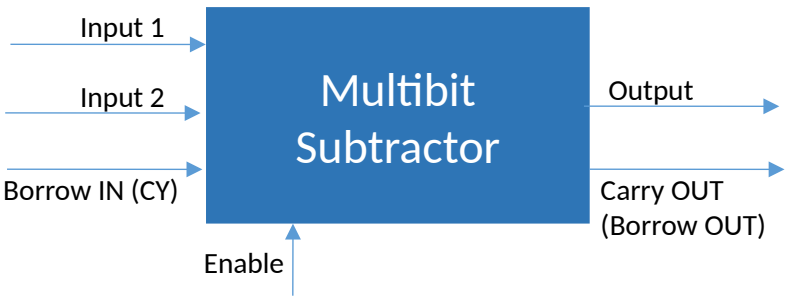




- Booth's Multiplier

Adders and Subtractors

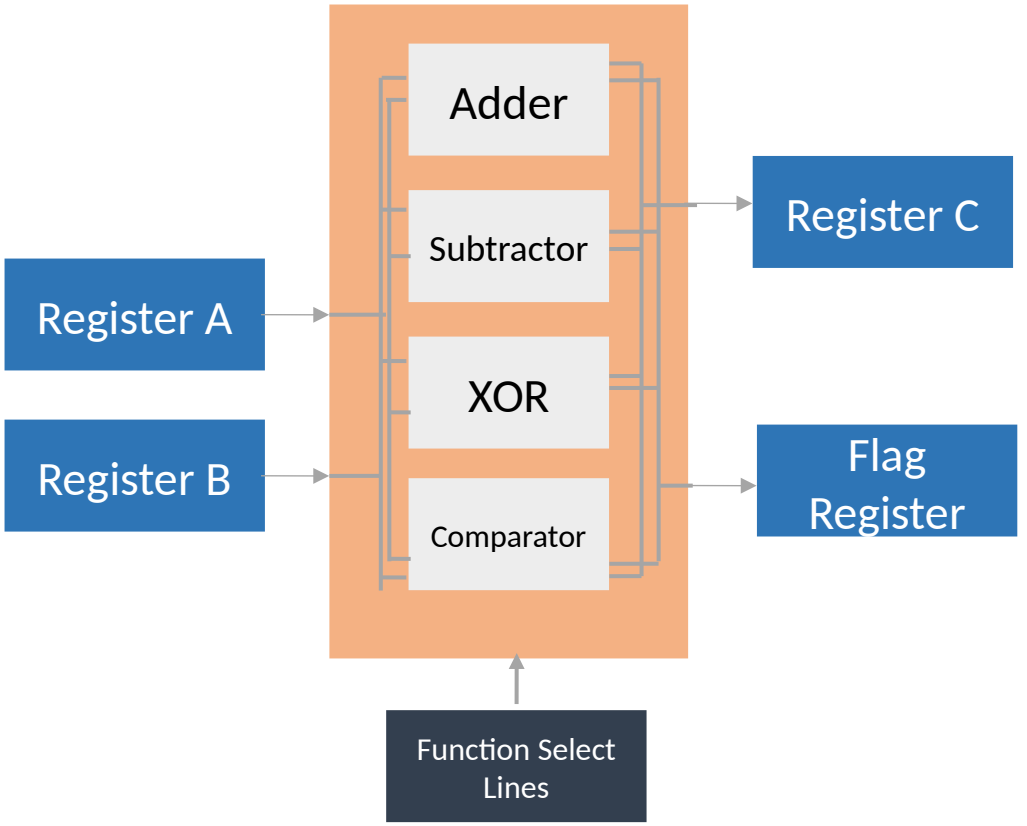
- Combinational Logic can be used to design multi-bit parallel Adders and Subtractors
  - Two multi-bit input words can be ADDED or subtracted giving an output word
  - Carry (CY) or Borrow (B) bits associated with adders and subtractors respectively: denoted here as FLAG bits



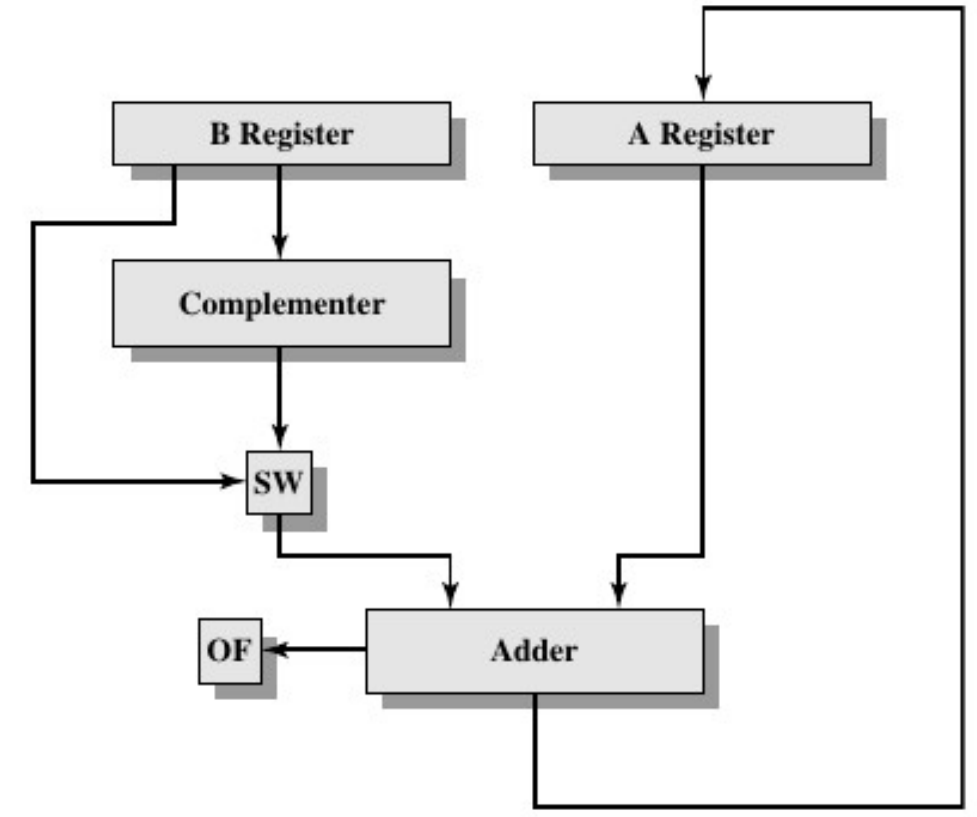
- Similarly one can design multi-bit Logic functions, Comparators, Complement functions etc.

Hardware Realization of ALU

- Execution Unit (EU) consists of ALU & registers among others
  - Registers are small memory units internal to EU
    - to hold temporarily the operands apart from the intermediate results in a computation
    - to hold the output and status of the result in a computation
  - Hardware computational units (design choices):
    - Design option #1: For every arithmetic & logical computations, design an exclusive hardware circuitry e.g. adder, comparator, XOR, subtractor, etc.
    - Design option #2: Design a common combinational circuitry which implements each of the above operations with an option (through function selection lines or control bits) to choose any one of them at a time.
    - In early 70's design option #2 was adopted in many commercial CPUs eg. ALU 74181
      - Advantage is that a single circuitry is reused for all possible ALU operations --> optimizes hardware, reduces footprint etc.
- ALU Function Select lines enable one of these units to perform, at a time
  - Input from and output to a set of pre-defined registers on the bus
  - Flag register contain bits for CY, B, Comparator result (Y/N), Zero etc.



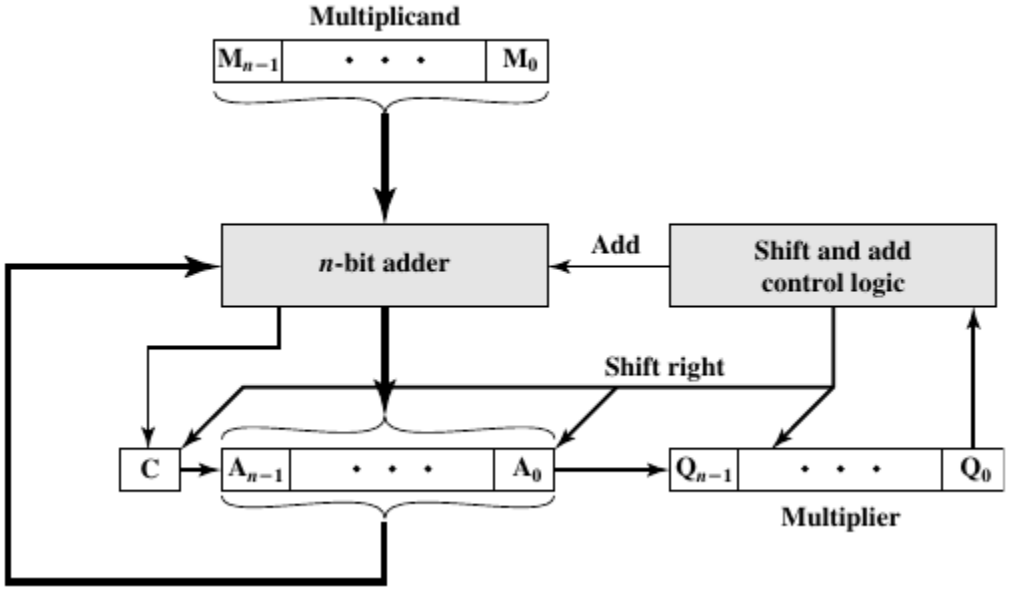
Hardware for Addition & Subtraction



OF = Overflow bit
SW = Switch (select addition or subtraction)

Unsigned Binary Multiplication

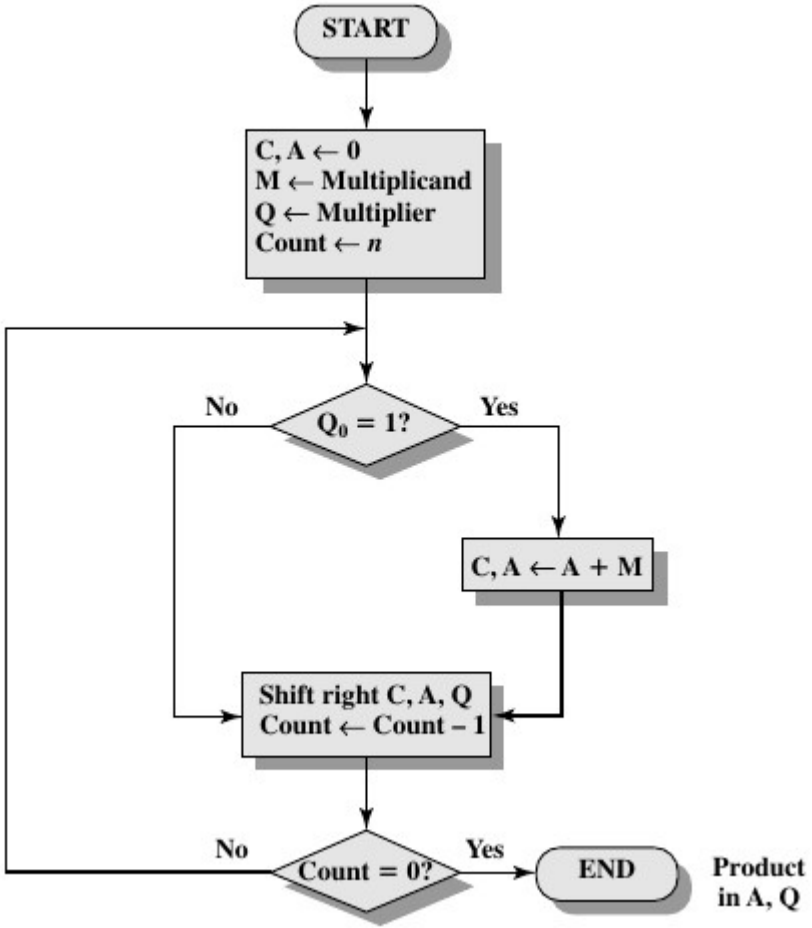
- Digital Circuit of Conventional Shift & Add Multiplier
- Improved version
  - Use of accumulator reduces memory space required
    - Save time in computation of partial products
    - 1 --> multiplicand
    - 0 --> zero
- Only unsigned binary multiplication



C	A	Q	M		
0	0000	1101	1011	Initial values	
0	1011	1101	1011	Add	First cycle
0	0101	1110	1011	Shift	
0	0010	1111	1011	Shift	Second cycle
0	1101	1111	1011	Add	Third cycle
0	0110	1111	1011	Shift	
1	0001	1111	1011	Add	Fourth cycle
0	1000	1111	1011	Shift	

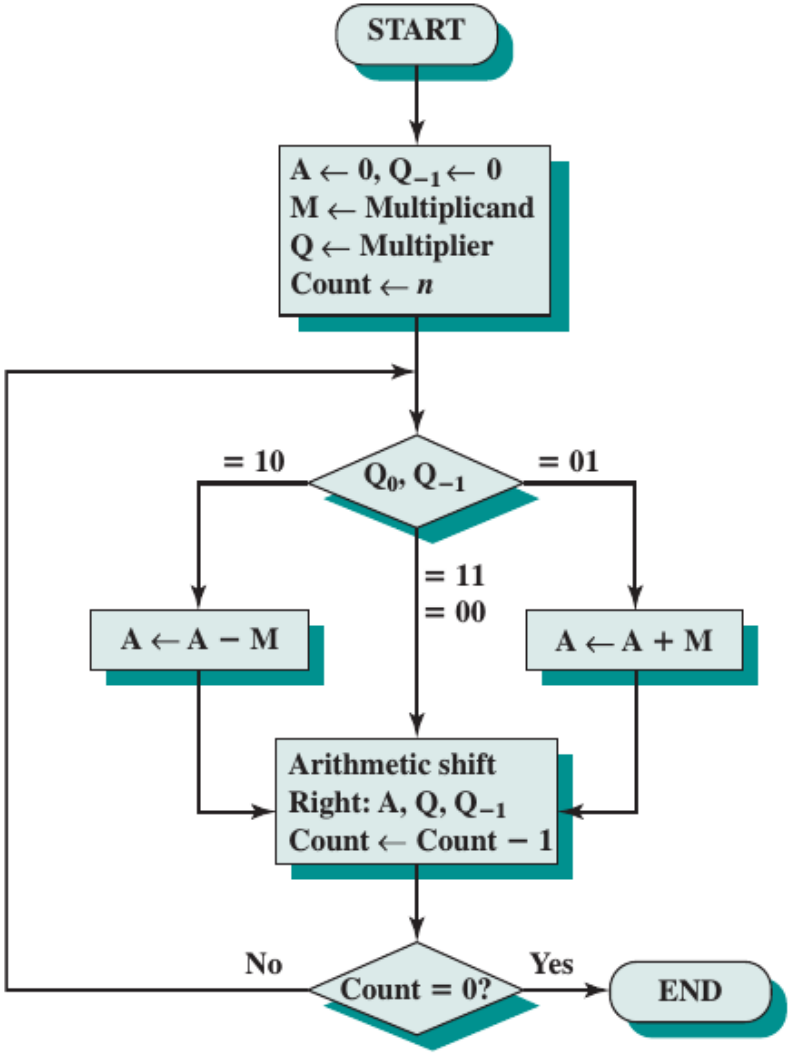
Flow chart for unsigned binary multiplication

- Flow chart for unsigned binary multiplication
  - Only unsigned numbers
- Use of 2s compliment?
  - Problems
  - If both multiplicand and multiplier are negative
  - Either of them negative --> fails
- S



Booth’s Algorithm

- Booth’s algorithm yields minimum number of computations (minimum computational overhead)
- S



Performance of Booth’s Algorithm

- Given a multiplication problem, time taken to compute the product in various algorithms are compared
- S

S. No		Multiplier Algorithm	Delay time
1		Conventional Array	92 ns
2		Vedic	39 ns
3		QSD	33.891 ns
4		Modified Booth’s algoritm	11.525 ns

Function Select Lines

- Each operation that an ALU can perform has a corresponding function line
  - Number of function lines = number of arithmetic /logic operations that a processor can perform
- Each of these lines activate one of the functional units inside the ALU
  - processor designer decides which function line triggers which operation
- Function lines activated/enabled by control unit of the processor

Function Line Number	Function
0	Add values in R1, R2
1	Increment value in R1
2	Compare Values in R1, R2
3	AND values in R1, R2
4	XOR values in R1, R2
5	Negate Value in R1
6	Subtract R1 from R2

Flag / Status Register

- A flag register or Condition Code Register is a collection of status flag bits giving information about the state of last operation performed (after operation)
  - Different bits in register acts as a separate status indicators (defined by processor designer)
  - Flag bits can be used as input in ALU operation (eg: Add with Carry) as well as input in an instruction (eg: Jump if Carry)

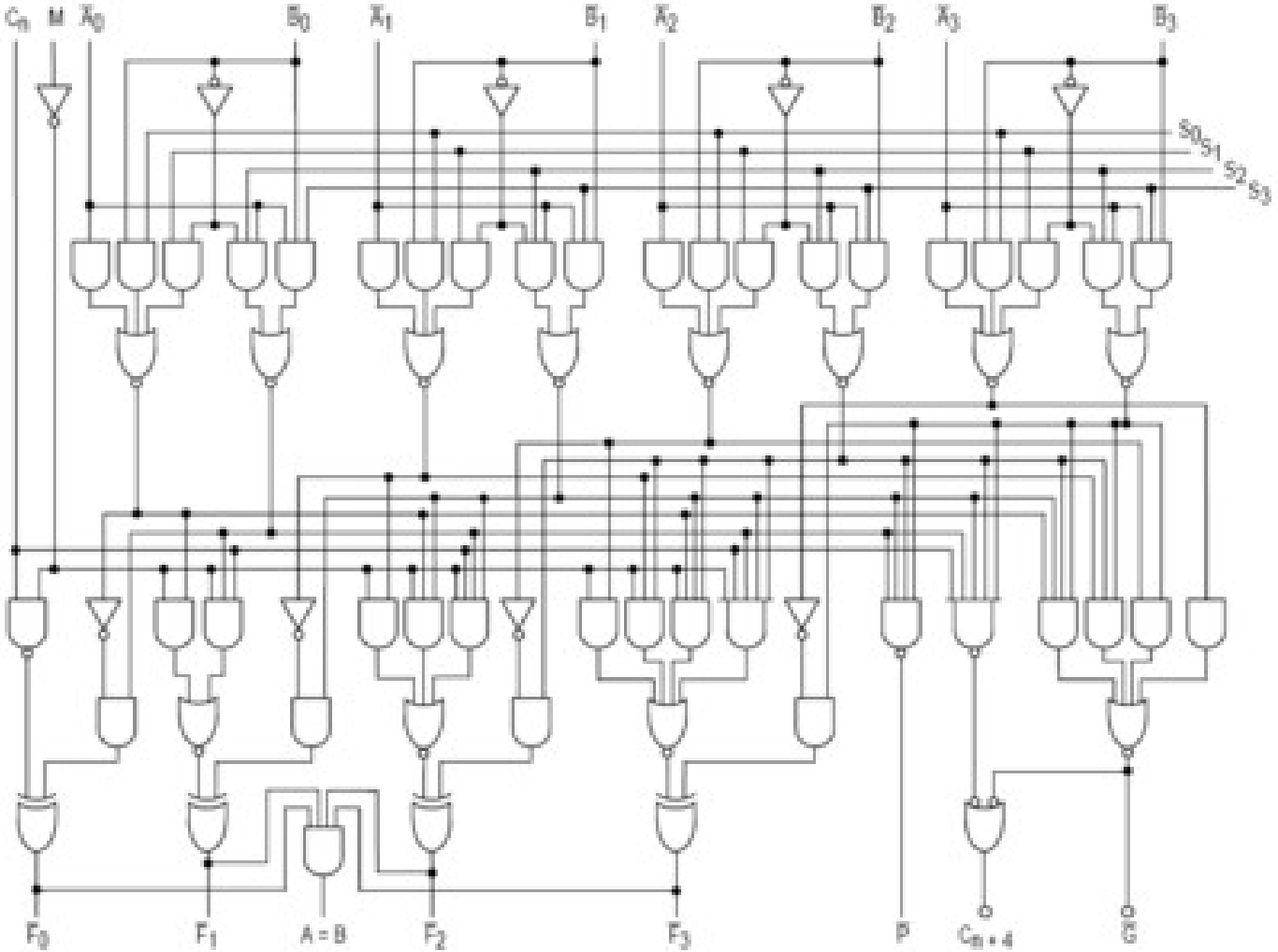
Flag Bit	Flag	Flag Name	Status Description
0	**Z**	Zero Flag	Indicates that the result of an arithmetic or logical operation (or, sometimes, a load) was zero.
1	**C**	Carry flag	Indicates whether carry was generated in previous addition
2	**N**	Negative flag	Indicates that the result of a mathematical operation is negative
3	**V / O / W**	Overflow flag	Indicates that the signed result of an operation is too large to fit in the register width using two's complement representation.

Mapped Registers

- Some processor registers mapped to serve a particular role for ALU
  - Mapped Input Registers: These are the input registers that are mapped to act as the Inputs to the ALU functional units
  - Mapped Output Register: Register that is used to always store the output of the ALU operation is called Mapped output register
  - Flag Register: Similarly, a register that is used by ALU to reflect the flag values generated after performing an operation is called flag register
- These registers are hard wired to function as desired role
  - The processor designer assigns these roles to the register
  - Eg. R0, R1 -> Input registers, R2 -> Output register, R3 -> Flag Register
- Modern processors dont have mapped registers (all general purpose registers (GPRs))

4-bit ALU: 74181 integrated circuit

- The 74181
  - an evolutionary step between CPUs of the 1960s (discrete logic gates), and today's single-chip uP CPUs.
  - Although no longer used in commercial products, it is still referenced textbooks & technical papers.
  - Sometimes used in "hands-on" college courses for to train future computer architects.
- The 4-bit wide ALU can perform
  - add / subtract / decrement operations with or without carry, as well as AND / NAND, OR / NOR, XOR, and shift.
  - Multiply and divide functions are not provided
- ALU Function Select lines enable one of these units to perform specific operation
  - S0, S1, S2, S3
  - Ref: wiki

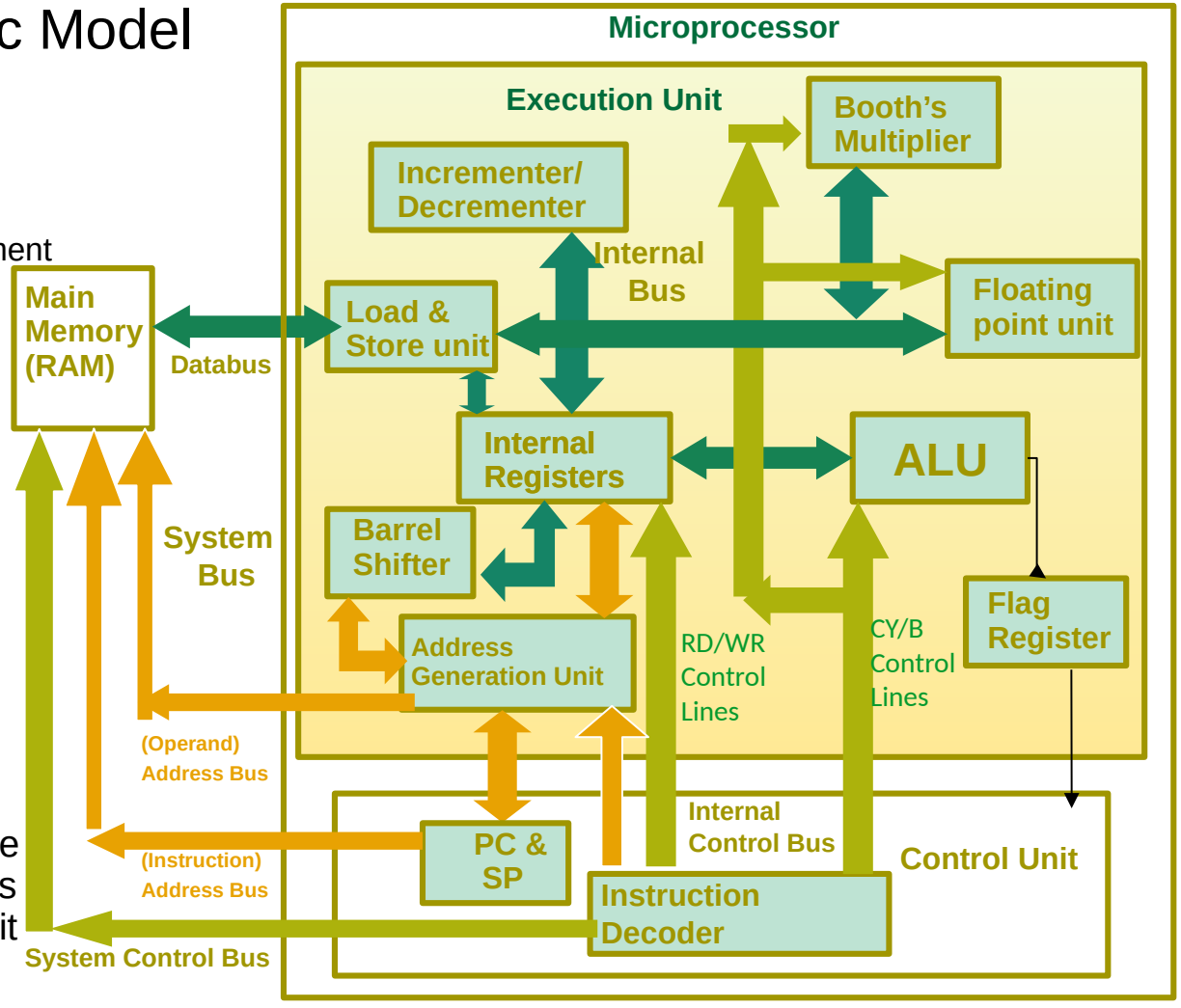


Function Table of 74181 4-bit ALU

Selection				Active-low inputs & outputs		Active-high inputs & outputs	
S3	S2	S1	S0	Logic (M = 1)	Arithmetic (M = 0) (Cn = 0)	Logic (M = 1)	Arithmetic (M = 0) (Cn = 1)
0	0	0	0	$\overline{A}$	A minus 1	$\overline{A}$	A
0	0	0	1	$\overline{AB}$	AB minus 1	$\overline{A+B}$	$A+B$
0	0	1	0	$\overline{A}+B$	$A\overline{B}$ minus 1	$\overline{A}B$	$A+\overline{B}$
0	0	1	1	Logical 1	-1	Logical 0	-1
0	1	0	0	$\overline{A+B}$	A plus $(A+\overline{B})$	$\overline{AB}$	A plus $(A\overline{B})$
0	1	0	1	$\overline{B}$	AB plus $(A+\overline{B})$	$\overline{B}$	$(A+B)$ plus $(A\overline{B})$
0	1	1	0	$\overline{A\oplus B}$	A minus B minus 1	$A\oplus B$	A minus B
0	1	1	1	$A+\overline{B}$	$A+\overline{B}$	$A\overline{B}$	$A\overline{B}$ minus 1
1	0	0	0	$\overline{A}B$	A plus $(A+B)$	$\overline{A}+B$	A plus AB
1	0	0	1	$A\oplus B$	A plus B	$\overline{A\oplus B}$	A plus B
1	0	1	0	B	$A\overline{B}(A+B)$	B	$(A+\overline{B})$ plus AB
1	0	1	1	$A+B$	$A+B$	AB	AB minus 1
1	1	0	0	Logical 0	A plus A	Logical 1	A plus A
1	1	0	1	$A\overline{B}$	AB plus A	$A+\overline{B}$	$(A+B)$ plus A
1	1	1	0	AB	$A\overline{B}$ plus A	$A+B$	$(A+\overline{B})$ plus A
1	1	1	1	A	A	A	A minus 1

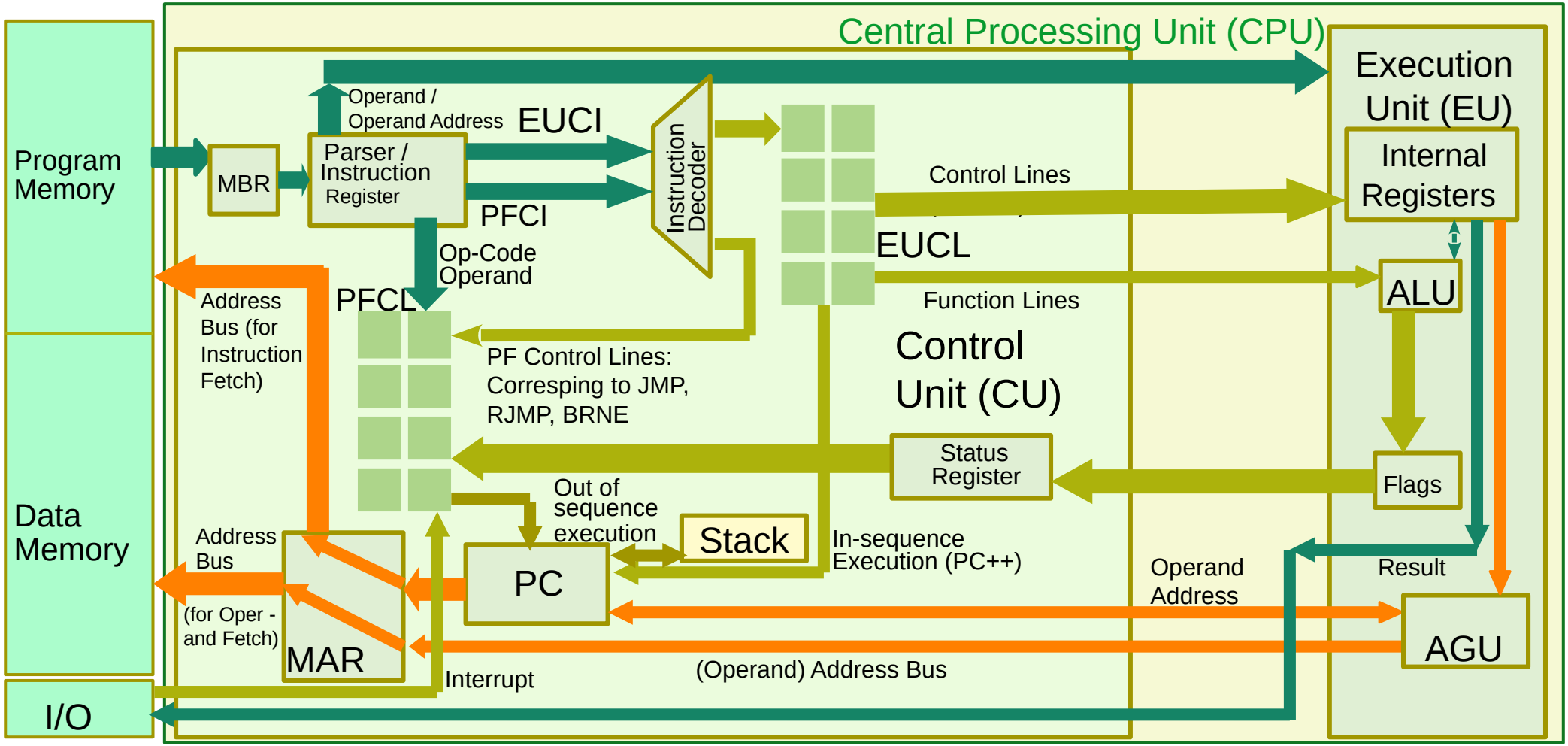
Engines in Execution Unit: Generic Model

- ALU
  - Arithmetic Operations
    - ADD, ADD with carry, SUB, SUB with borrow, 2's complement, 1's complement, increment / decrement
  - Logical Operations
    - AND, OR, Ex-OR
- Shifting (Barrel shifter)
  - Arithmetic shift
  - Logical shift
  - Rotate
  - Rotate through carry
- Address Generation Unit
  - Calculates the addresses in main memory as requested by control unit
  - Those address-generation calculations involve different integer arithmetic operations, such as addition, subtraction, modulo operations, or bit shifts.



Engines in Execution Unit: Generic Model

- Load & Store Unit
  - In LSA architecture, instructions exclusively do either memory or ALU operations
  - No ALU operations possible directly on operands from main memory, in LSA
  - The load–store unit (LSU) usually includes a queue which acts as a waiting area for memory instructions, and the unit itself operates independently of other processor units.
- Internal General Purpose Registers (GPRs)
  - Within EU, faster & scratchpad purpose.
- Special Function Registers (SFRs)
  - PC (Program Counters), SP (Stack Pointers), LR (Link Register), SR (Status Registers)
- Booths Multiplier
  - Multiplies two signed 2's complement numbers
  - Overall number of computations (binary additions, subtractions, shift operations) in Booths algorithm are less than that of conventional multiply-shift-add algorithm --> faster
- Incrementor/ Decrementer
  - Incrementer / Decrementer is used for calculating memory addresses, particularly for array indexing or sequential data access
  - PC update, loop control (eg analogous to "for" loops),
  - Pointer manipulations (increment / decrement pointer values)
- Flag Registers
  - Result of arithmetic and logical operation in EU is often accompanied by events (flags) which are used in furtherance of calculations
  - Typical flags are: Negative (N), Zero (Z), Sign (S), Carry (C)
- Floating Point Unit
  - Computations of all of floating point arithmetic


Central Processing Unit (CPU)
Program Memory
Data Memory
I/O
MBR
Parser / Instruction Register
Operand / Operand Address
EUCI
PFCI
Op-Code Operand
PFCL
Instruction Decoder
Control Lines
EUCL
Function Lines
Control Unit (CU)
Execution Unit (EU)
Internal Registers
ALU
Flags
Status Register
PF Control Lines: Corresping to JMP, RJMP, BRNE
Address Bus (for Instruction Fetch)
Out of sequence execution
Stack
In-sequence Execution (PC++)
PC
Address Bus
(for Oper - and Fetch)
MAR
Interrupt
(Operand) Address Bus
Operand Address
Result
AGU

Putting CU Together: Control Unit

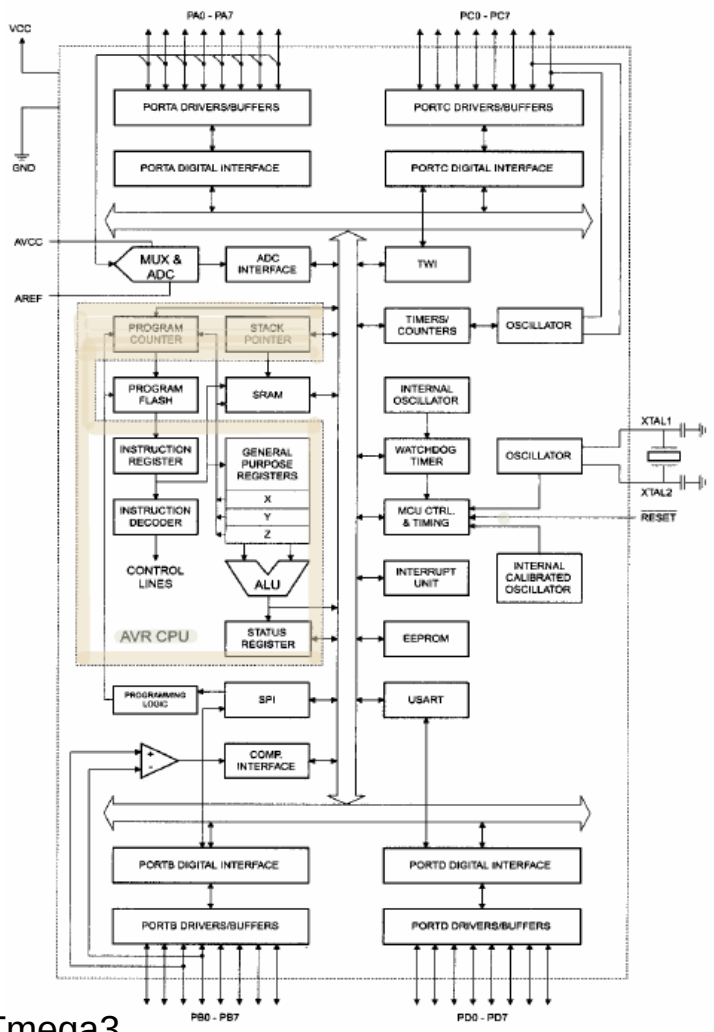
- Memory Buffer Register (MBR)
  - Memory which stores temporarily the instruction to be executed, currently
- Memory Address Register (MAR)
  - Register which contains the address of the operand to be fetched
- Instruction Register (or Instruction Decoder)
  - Decodes the given OP code and generates hardware level signals to execute the corresponding command
- (Call) Stack & Stack Pointer
  - Special memory area (stack data structure), wherein the information about the active subroutines are stored.
- Program Counter (PC)
  - Instruction pointer – points to the next instruction to be executed
- Other Internal registers in CU
  - Flag register (eg. CPSR) and status registers (eg. Program Status Word)

Putting CU Together: Control Unit

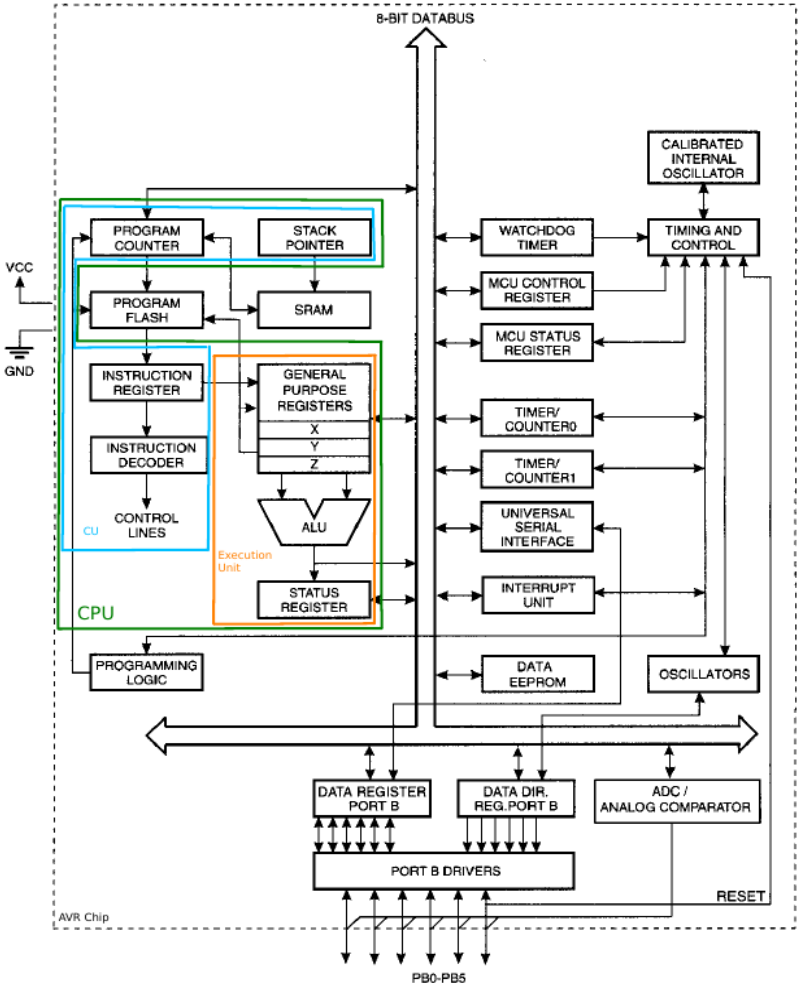
- Execution Unit Control Logic (EUCL):
  - generates the necessary control signals (once the instruction is decoded) that activate specific functional units within the EU.
  - The above control signals coordinate the activities of the ALU, registers, shifters, AGC & other components to execute the instruction correctly.
  - manages the allocation of hardware resources within the EU to ensure that the instruction can be executed without conflicts – eg. ensure that registers are available for reading and writing, that the ALU is free for the required operation, and that data paths are correctly configured.
  - coordinates the timing and synchronization of operations within the EU, ensuring that instruct-ions are executed in the correct sequence and that results are available when needed.
-

- Program Flow Control Logic (PFCL):
  - responsible for coordinating and controlling the operations, including decoding and executing PFCI instructions
  - handles control flow instructions, such as branches & (conditional) jumps.
  - manages conditional execution instructions, evaluates condition flags or values to decide whether to execute the instruction or skip it - essential for implementing conditional branching
  - determines the new program counter (PC) value to specify the address of the next instruction to be executed, during branches & (conditional) jumps.

AVR Microcontrollers



ATmega32



ATiny25