Computer architecture

Elements of control unit (CU)

von Neumann computer

- Program to be executed be stored in binary format in a memory device so that intermediate results could cause the desired effect to the program.
- This is also known as the Stored Program Concept in Computer design
- The stored program concept says that the program i.e. instructions are stored along with the data in the computer’s memory in machine-readable binary form(machine language)

von Neumann architecture

- This execution model needs to have certain units, the basic building blocks of von Neumann computer:
  - ALU
  - Memory
  - Input/output
  - Control unit
- ALU has to have a register called Accumulator
- Control unit has to have a register/counter called PC (*Program Counter* – it exists to follow which is the next instruction
- These registers are usually implemented as memory components in CPU, so that any intermediate result could cause the desired effect to the program. This is also known as the Stored Program Concept in Computer design.

How does von Neumann computer work?

- Executes or emulates Fetch-Decode-Execude sequence for program execution:
    - *Fetch* the instruction from memory at the address denoted by the Program Counter.
    - Increment the program counter to point to the next instruction to be fetched.
    - *Decode* the instruction using the control unit.
    - *Execute* the decoded instruction using the data path, control unit and ALU. As part of this step, any data that is to be fetched from memory for executing the instruction is brought. Also, if any results to be updated in memory is written into memory, at the end of the execution of an instruction.
- Go back to beginning after this sequence

The role of ALU

- Common ALU operations:
  - Arithmetic operations (+, -, …)
  - Logical operations (AND, OR, …)
  - Data movement operatins (Load, Store)
- ALU name comes from the basic operations that it does
- We implement ALU by using combination of different types of circuits

What does a computer startup look like?

- CPU and internal components reset to a set of pre-defined values
- PC register is loaded with memory location of the first instruction – depending on CPU architecture and BIOS programming
- PC is **very** important part of CU as modern computer cannot exist without it – PC contains the next instruction address
- At the start of execution cycle, CU reads information from address given by PC and loads it to an internal decoding and execution register – first word of the instruction is called **opcode**
- Opcode is important as that CU uses that word to learn does it need to load some additional data from additional memory locations, such as memory location or an operand

Control unit roles

- It’s a synchronous, sequential digital circuit
- It coordinates sequential data and instruction access to and from CPU and CPU registers
- Interprets CPU instructions and controls their execution
- As a part of the execution cycle, it cooperates with other CPU functional units and external devices (memory, U/I devices)
- It’s a part of instruction decoding process – based on that, CU identifies the instruction category (for example – branching or no branching instructions)

So, altogether, what does a CU do?

- Generates control signals
- Coordinates all CPU activities
- Synchronizes data transfer and module communication
- Fetches, decodes, and enables instruction execution
- Communicates with other components via buses (address, data, control)
- Manages external signal responses (interrupt, halt, ….)

Basic CU functions

1. Establishing a specific state during each instruction cycle
2. Selecting next state based on the current state, flags state in the status register and state at CPU control inputs
3. Storing information that describes the current CPU state
4. Generates control signals for data transfer between a CPU and other functional units

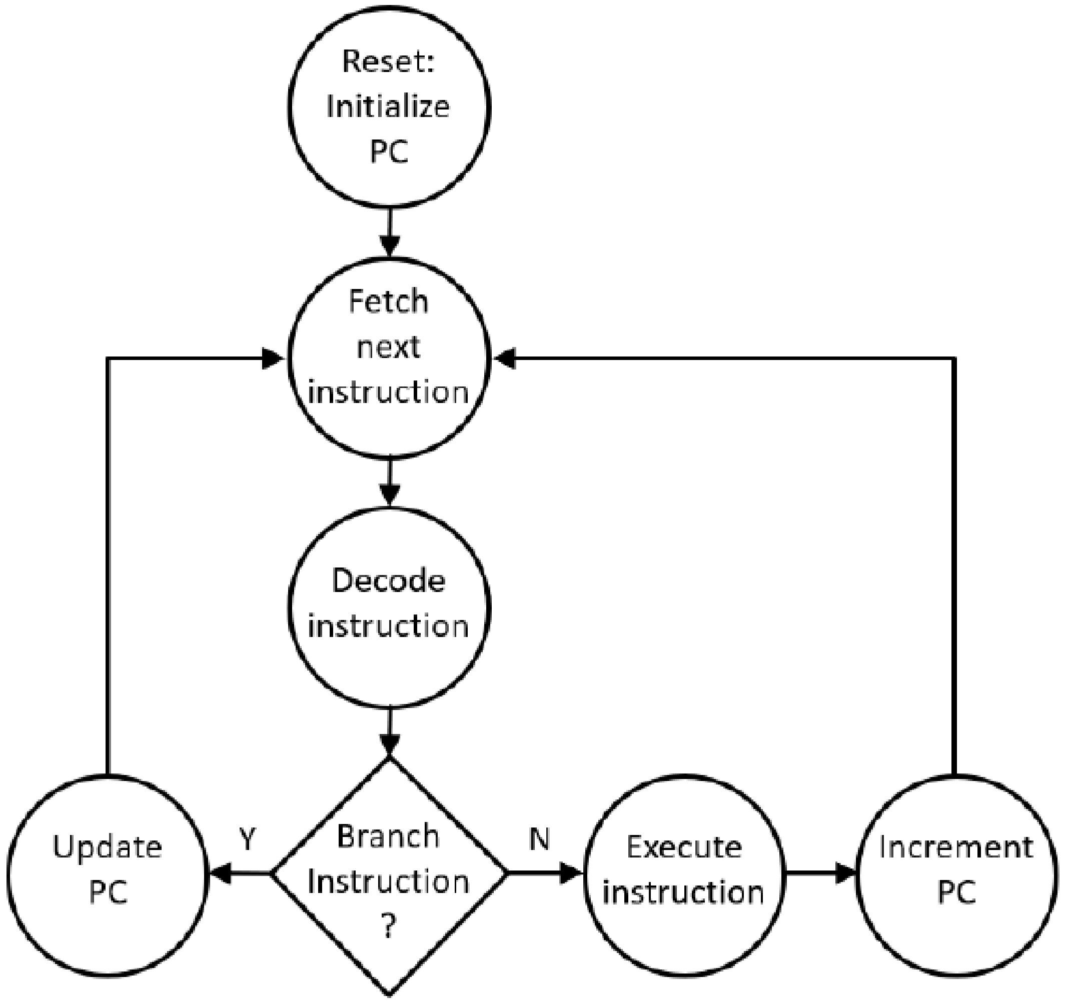
Transfer of control between applications

- 1. Programming procedures are the basis for structured programming
  - Procedure calls are used to transfer control flow from one program to another
  - When a task that was defined by a linked procedure (subprogram) is done, control has to go back to the instruction that’s the first next instruction to the instruction that called the procedure
- 2. Second, very important use case for transfer of control happens as a part of interrupt process
  - In this case, we’re talking about:
    - Interrupted program – a program whose execution ended with an interrupt
    - Interrupt program – program that gets the control, and usually serves some input/output unit that generated the interrupt

Transfer of control between applications

- 3. Transfer of control can also happen as a result of an exception
  - By exception, we’re talking about special situations that stop the normal program flow and execution of the current flow
  - Exceptions can be treated as interrupt subset
  - Exceptions can happen for two main reasons:
    - External exceptions – caused by something outside of the CPU
    - Internal exceptions – caused by something that happened inside the CPU

What does an instruction execution process look like?



How does a CU work?

- Converts input into control signals
- Control signals are needed so that instructions can be executed properly, or to handle interrupts, exceptions etc.
- Control signals are sent to the CPU
- Control signals are also used for communication with busses, memory and U/I systems
- CPU “tells” the available hardware devices which operations need to be executed

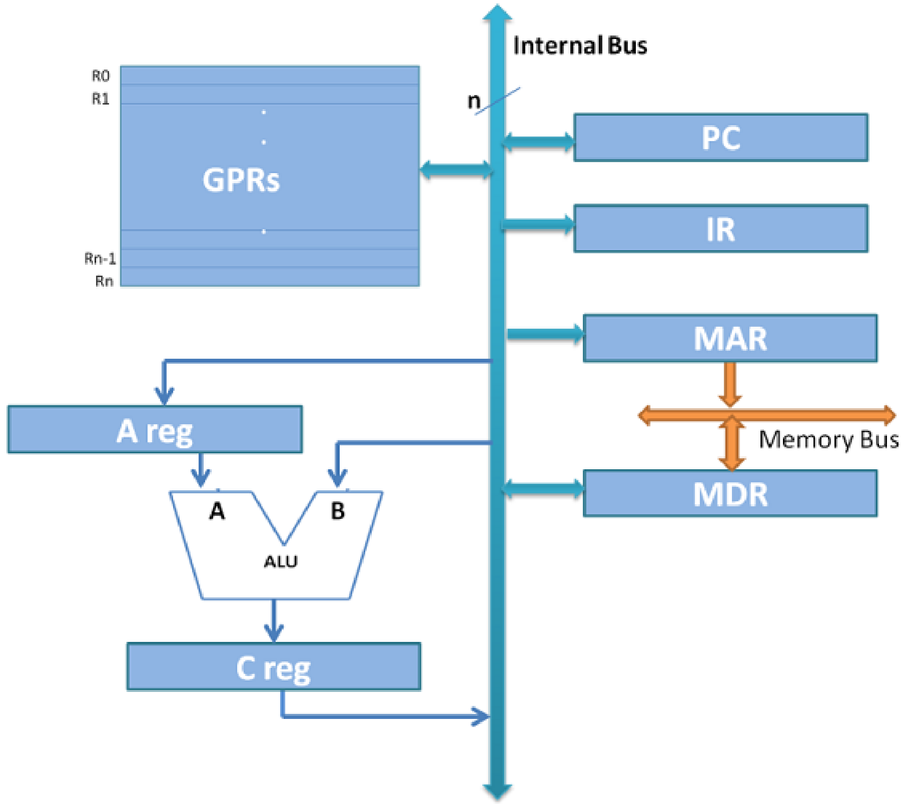
Devices that are a part of the execution cycle – I

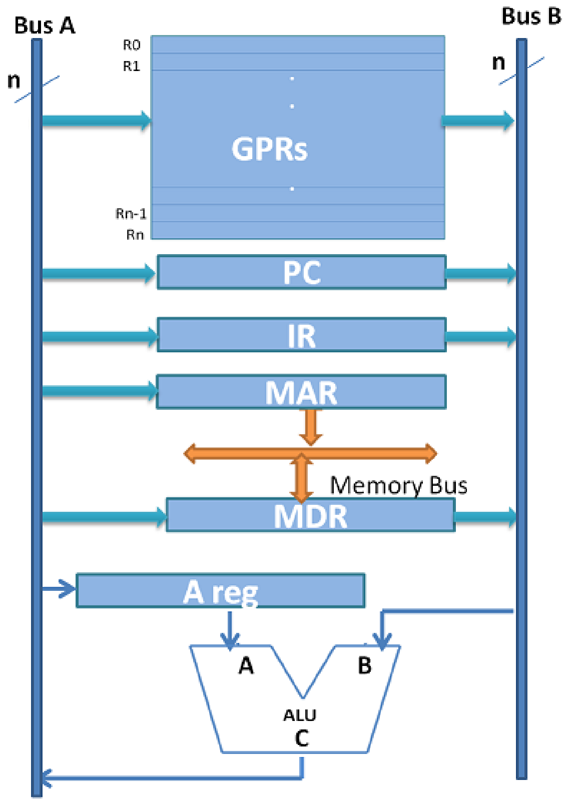
- ALU – works with data, arithmetic, address calculation
- Instruction register and decoder – instruction decoding (for the instruction that needs to be executed)
- PC – points to the next instruction in the flow, a part of the execution flow
- Memory
  - Instruction memory – in FETCH phase, almost always read-only
  - Data memory – needed to fetch operands and store results
  - MAR (Memory Address Register) – contains location of memory that needs to be accessed
  - MDR (Memory Data Register) – contains location of memory that holds the data that needs to be accessed

Devices that are a part of the execution cycle – II

- CPU registers – usually these are actually fast memory devices for various types of operations
- Internal registers, MUX’s, ….
- Internal bus – connects all of these elements
- CU – The Boss that handles all of the elements
- All of these elements need to be connected and work in sync
- Depending on the design, elements can be connected via one or multiple busses

Datapath with 1 or 2 busses

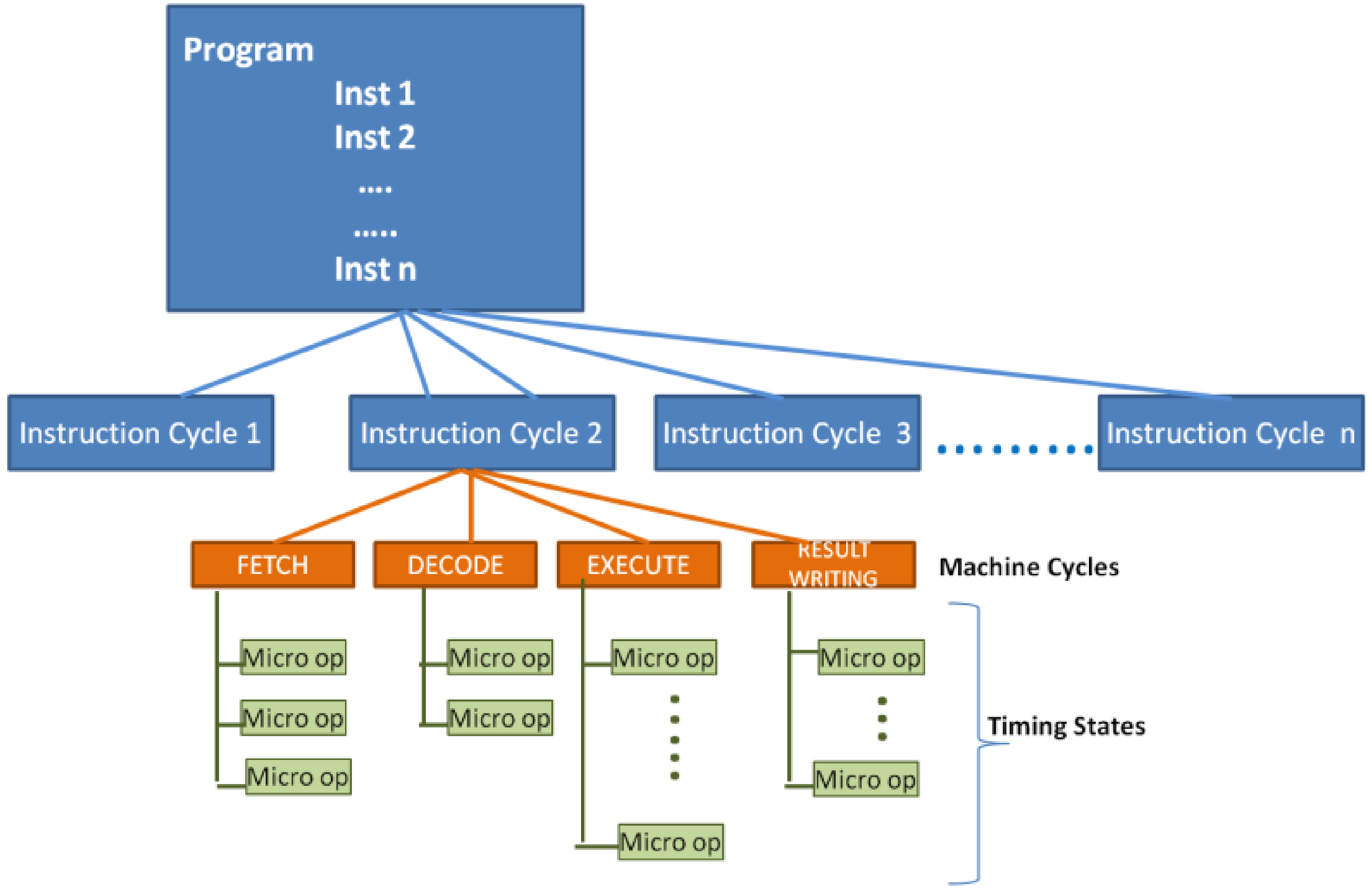




What does all of this mean from the standpoint of executing a program?

- Program contains a set of instructions
- Every instruction has macro-cycles, like FETCH, DECODE, EXECUTE and RESULT WRITING
- The amount of these cycles depends on CPU design – for example, some CPU architectures have built-in interrupt handling, some of them don’t
- Usually, we say that FETCH has two phases – fetching instruction, fetching operands for the instruction
- Every one of these macro-cycles has micro-steps, so called atomic operations
- Speed of execution is directly proportional to the CPU frequency

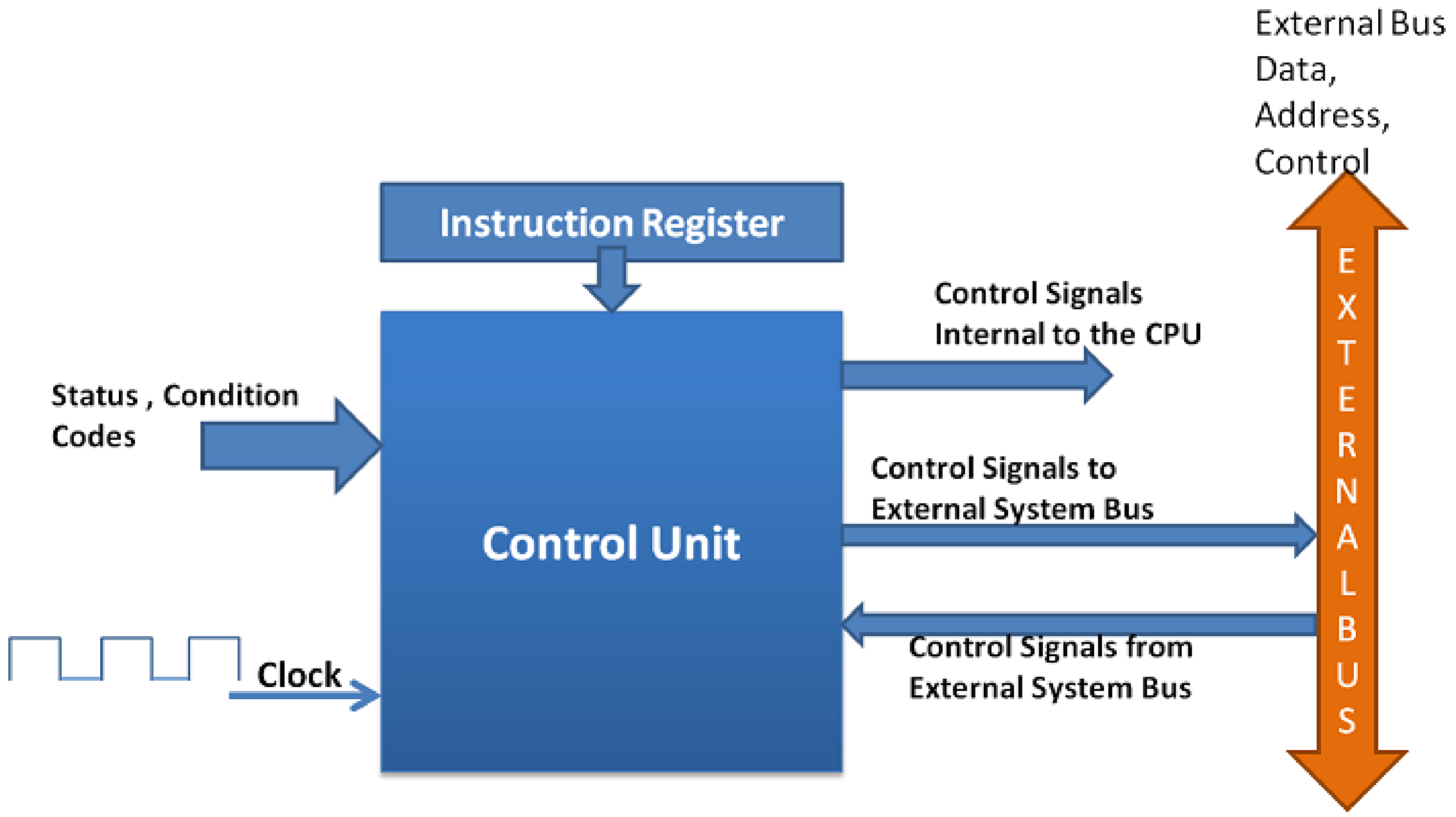
Typical program execution flow



CU – what is it, then?

- It’s a state machine that generates control states depending on input, while the output is generated based on conditions and input
- Important CU parameters:
  - CPU clock – it drives the CU, synchronizes micro-operations (one or more micro-ops per clock) – multiple operations per clock are possible if we’re using pipelining, parallelism or operations that aren’t in conflict with one another
  - Instruction register (IR) – contains the opcode for the current instruction, directly related to current micro-operations
  - Flags, control codes – flags like Trap, Interrupt, Parity – they depict the current CPU state and conditions, so that it knows the result of previously executed opretations
  - External control bus – it exists so that CPU can communicate with external devices – for example, I/O-based interrupt means that CPU needs to pay attention to it

Control unit model


External Bus
Data,
Address,
Control
Instruction Register
Control Signals Internal to the CPU
Status , Condition Codes
Control Unit
Control Signals to External System Bus
Control Signals from External System Bus
Clock
EXTERNAL BUS

Control unit signals

Output Control Signals from Control Unit

Internal to the CPU

- The signals that activate the ALU functions
- The signals that activate the Datapath for register to register data movement

External Control bus

- Control Signals like READ, WRITE, ADDRESS SYNC, etc to Memory
- Control signals to I/O subsystems like IOWRITE, IO READ, INT ACK, IOADDRESS SYNC etc.

Instruction types, from the CU standpoint

- General instructions – executed on the CPU directly, sequentially as they come
  - CU per von Neumann concept – instruction after instruction, serialized, rinse, repeat
- Branching instructions – these are implemented by the CU itself
  - When these instructions get executed, PC stores the memory address where branching starts
  - Instructions like conditional branching, unconditional branching (jump), sub-routine call, sub-routine return

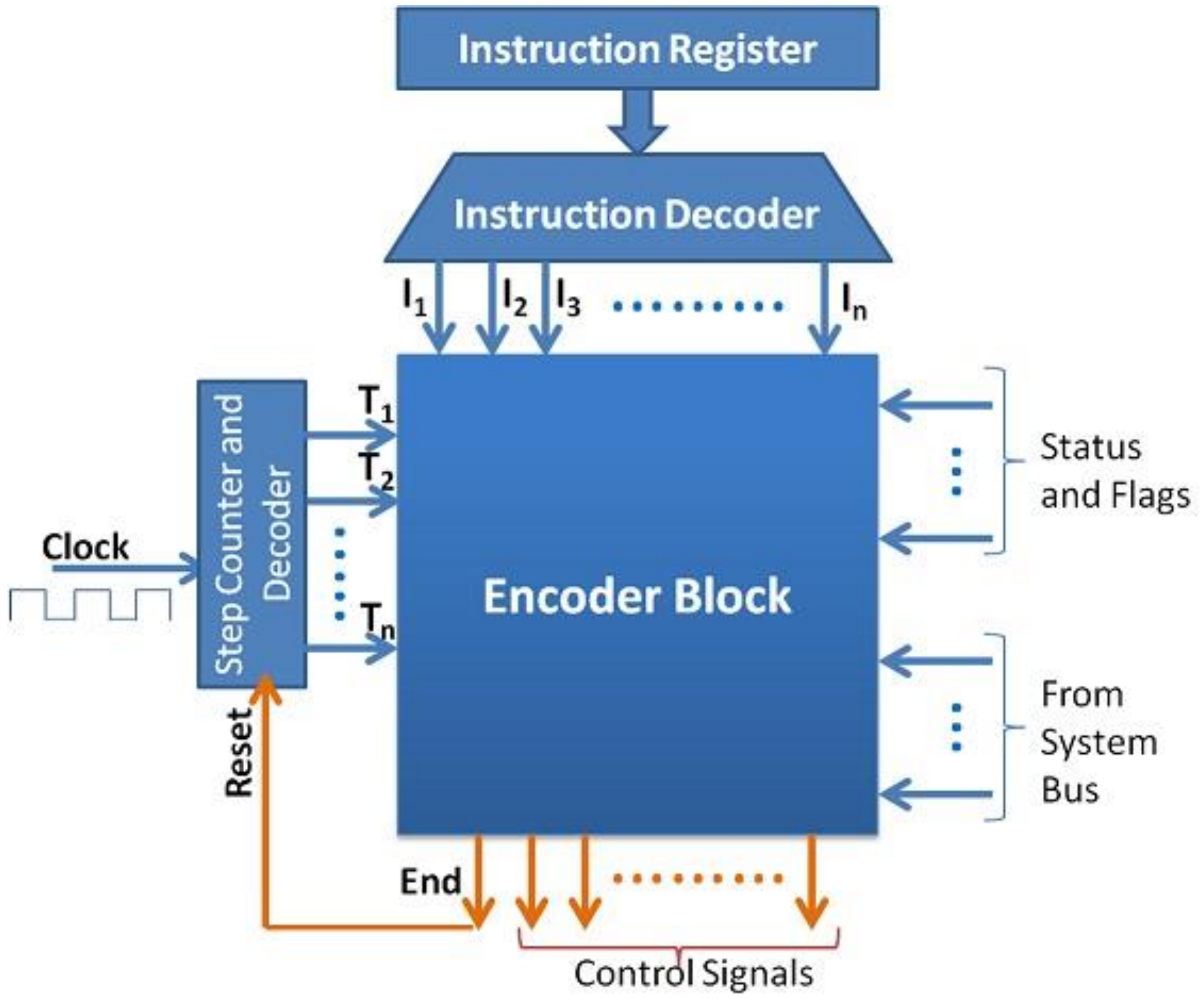
Design and CU types

- Usually, CU is designed after we analyze the datapath
- This means that we know which control signals are needed, which, by extension, means that we know what we need out of our CU design
- Two main tasks that CU does – sequencing (fetching data) and executing (by using available execution units)
- Two main logical approaches to CU design:
    - Hardwired
    - Micro-programmed

Hardwired CU

- Operation order depends on how the underlying logic circuits are connected inside the CU (hardwired)
- Logic uses gates, decoders, encoders, counters, directly connected to the CU
- Cons:
  - Limited functionality
  - Difficult to make, design, and test
  - Very difficult to add new instructions
  - If we're using this CU type for CISC CPU, it's way to limiting considering the sheer amount of instructions that CPU supports
- Pros:
  - Hardwired CPU is way faster than a micro-programmed CU
  - It's very popular for RISC-based CPUs – RISC CPUs are usually 1i/per clock, so this type of logic suits them just fine

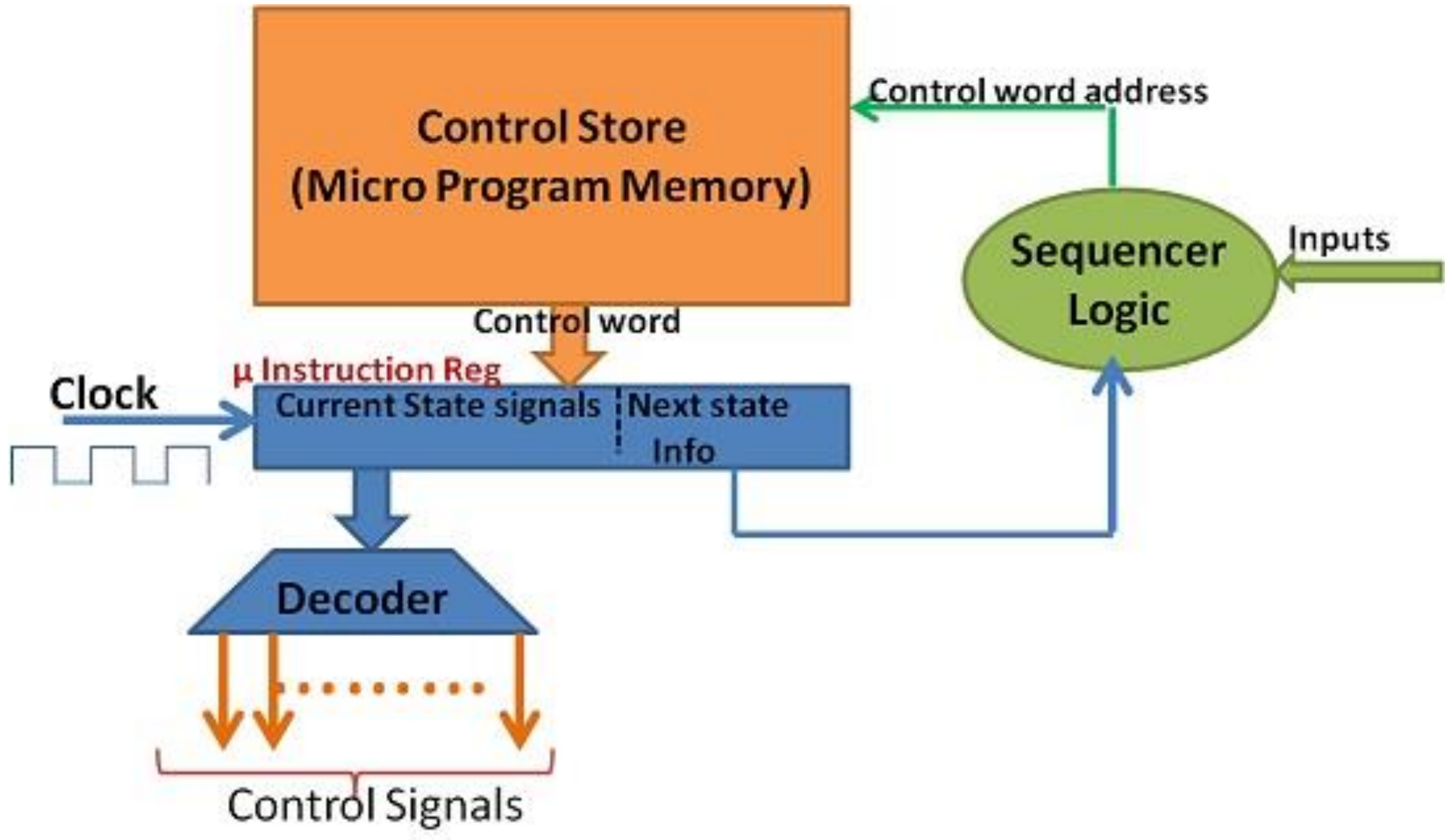
Typical *hardwired* CU



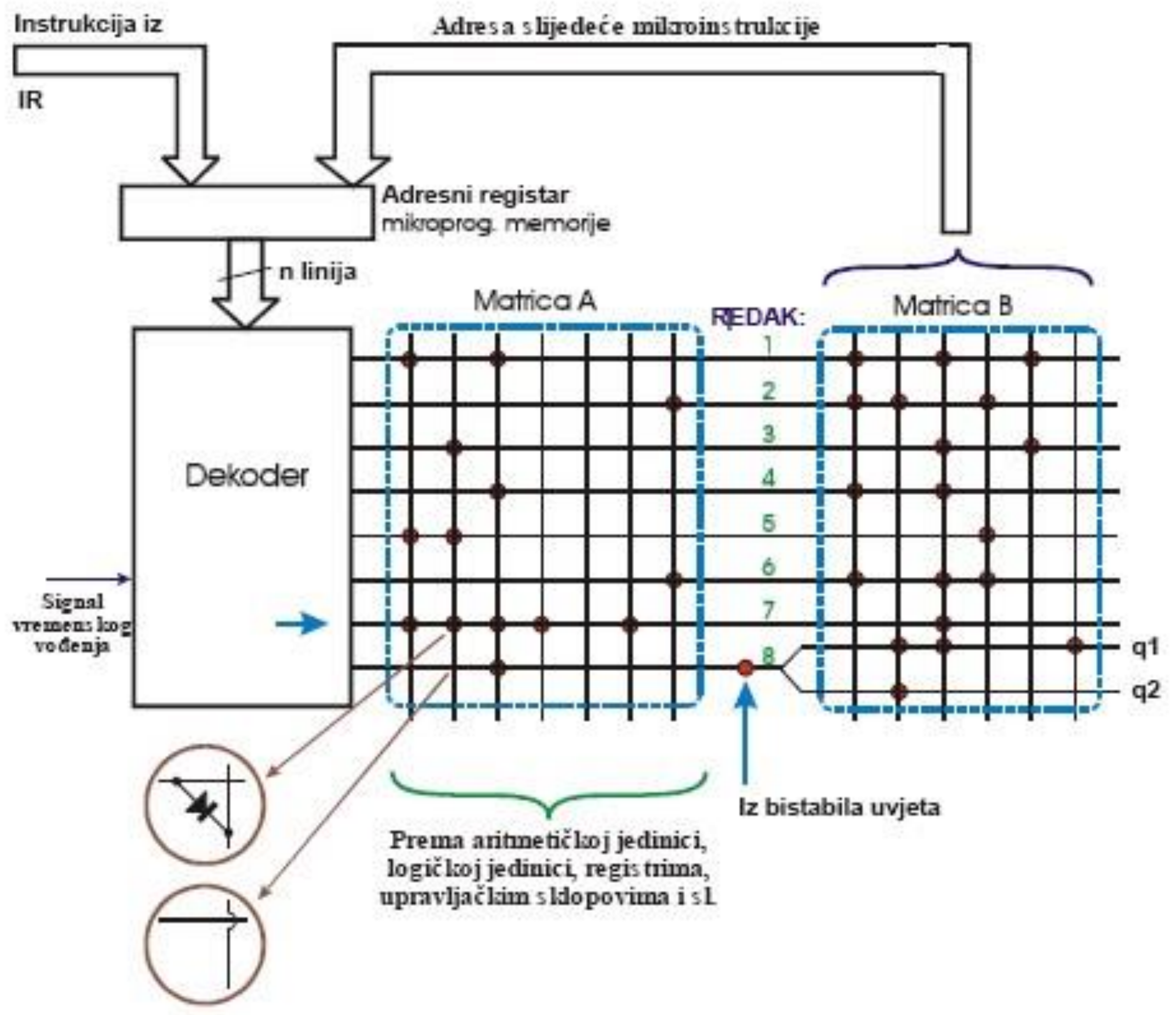
Micro-programmed CU logic

- Completely different design, based around memory concepts
- It’s often referred to as a memory-based CU
- Memory location content, after decoding, basically becomes the basis to generate control signals for execution
- Info about the next state is in memory, fetch, cycle starts again
- “computer within computer” design
- During execution of micro-program, CU goes through the following phases:
  - Microphase Fetch – fetching the micro-instruction
  - Microphase Execute – executing the micro-instruction

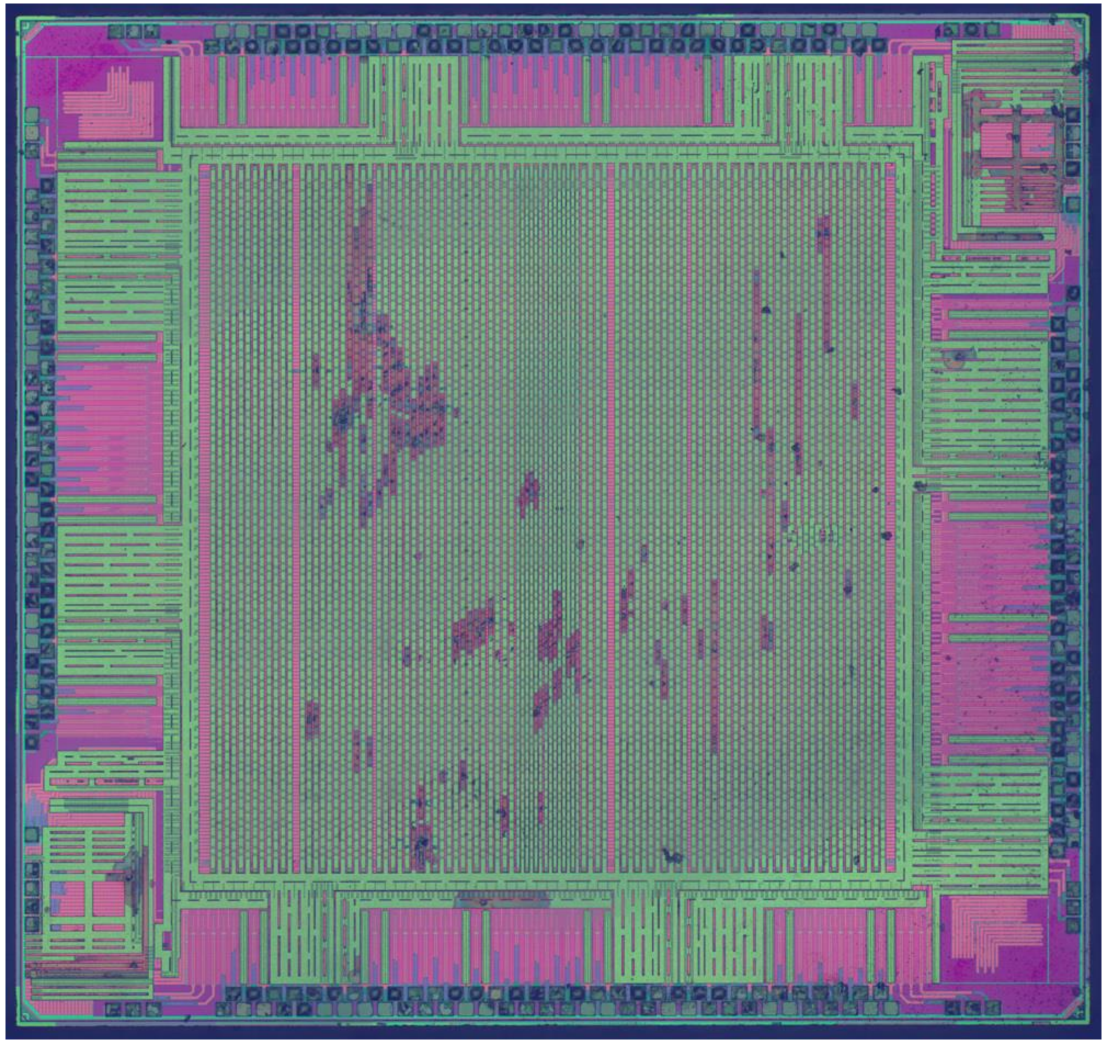
Typical *micro-programmed* CU



Wilkes scheme for micro-programmed CU


Instrukcija iz
IR
Adresa slijedeće mikroinstrukcije
Adresni registar mikroprog. memorije
n linija
Dekoder
Matrica A
REDAK:
Matrica B
1
2
3
4
5
6
7
8
q1
q2
Signal vremenskog vođenja
Iz bistabila uvjeta
Prema aritmetičkoj jedinici, logičkoj jedinici, registrima, upravljačkim sklopovima i sl.

Altera Cyclone V FPGA



Types of micro-programmed CU

- Single-level control store
- Two-level control store – besides the control store for micro-instructions, it also has a control store for nano-instructions
    - In this type of CU, nano-instruction control store knows of all control signal combinations
    - Using this type of design can have a direct impact on word length in micro-instructions, which can also mean that CU needs less memory

Pros et cons (micro-programmed CU)

- Pros:
  - Much simpler design and modification
  - Architecture and micro-code design can be done in parallel
  - CU problems can be solved by firmware upgrade
  - If it’s done properly, it’s usually more efficient with internal registers
  - Much more in line with the CISC way of thinking with the complex instruction set
- Cons:
  - Usually slower than a hardwired type
  - Usually more problematic with parallelism
  - Usually requires quite a bit of memory, which can be a design and price problem

Hvala na pažnji!