Introduction to Embedded Systems
EHB326E
Lectures

Prof. Dr. Müştak E. Yalçın

Istanbul Technical University

mustak.yalcin@itu.edu.tr

General-Purpose Processors: Software

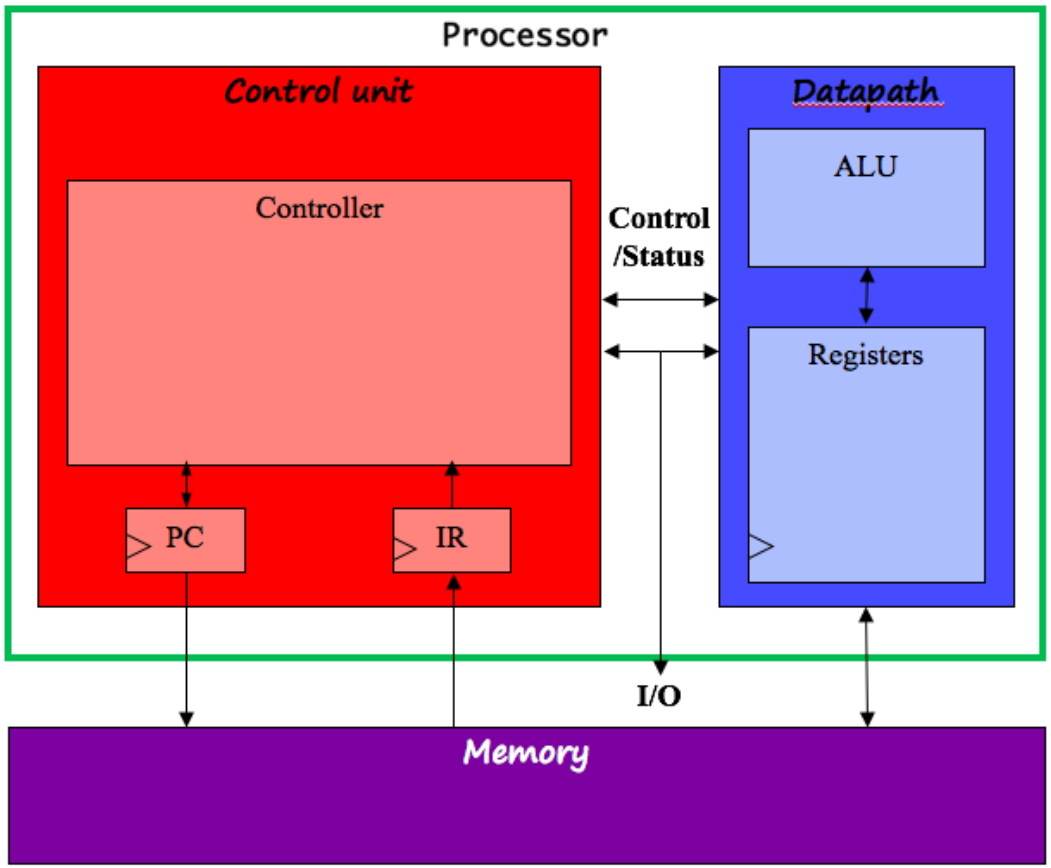
- Processor designed for a variety of computation tasks
- Low unit cost, in part because manufacturer spreads NRE over large numbers of units
  - Motorola sold half a billion 68HC05 microcontrollers in 1996 alone
- Carefully designed since higher NRE is acceptable
  - Can yield good performance, size and power
- Low NRE cost, short time-to-market/prototype, high flexibility
  - User just writes software; no processor design
- a.k.a. "microprocessor," "micro" used when they were implemented on one or a few chips rather than entire rooms

BLG 212E Microprocessor Systems

General-Purpose Processors: Basic Architecture

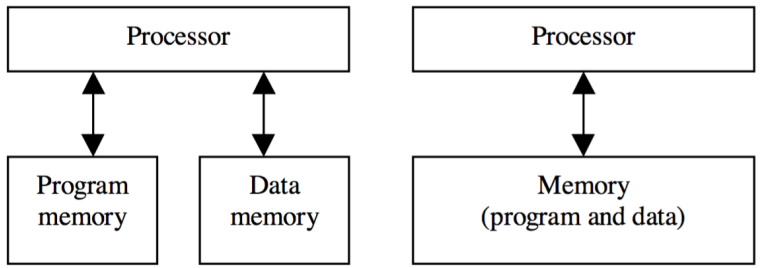
Key differences

- Datapath is general
- Control unit doesn't "store the algorithm" the algorithm is "programmed" into the memory



General-Purpose Processors: Basic Architecture

- Datapath Unit: consists of circuitry for transforming data and for storing temporary data.
  N-bit processor : N-bit ALU, registers, buses, memory data interface
- Control Unit: consists of circuitry for retrieving program instructions and for moving data to, from and through the datapath according to those instr.
  Program Counter's size determines address space
- Memory: While registers serve a processorÄôs short term storage requirements, memory serves the processorÄôs medium and long-term information-storage requirements. Two memory architectures: Harvard and Princeton.



General-Purpose Processors: Operations

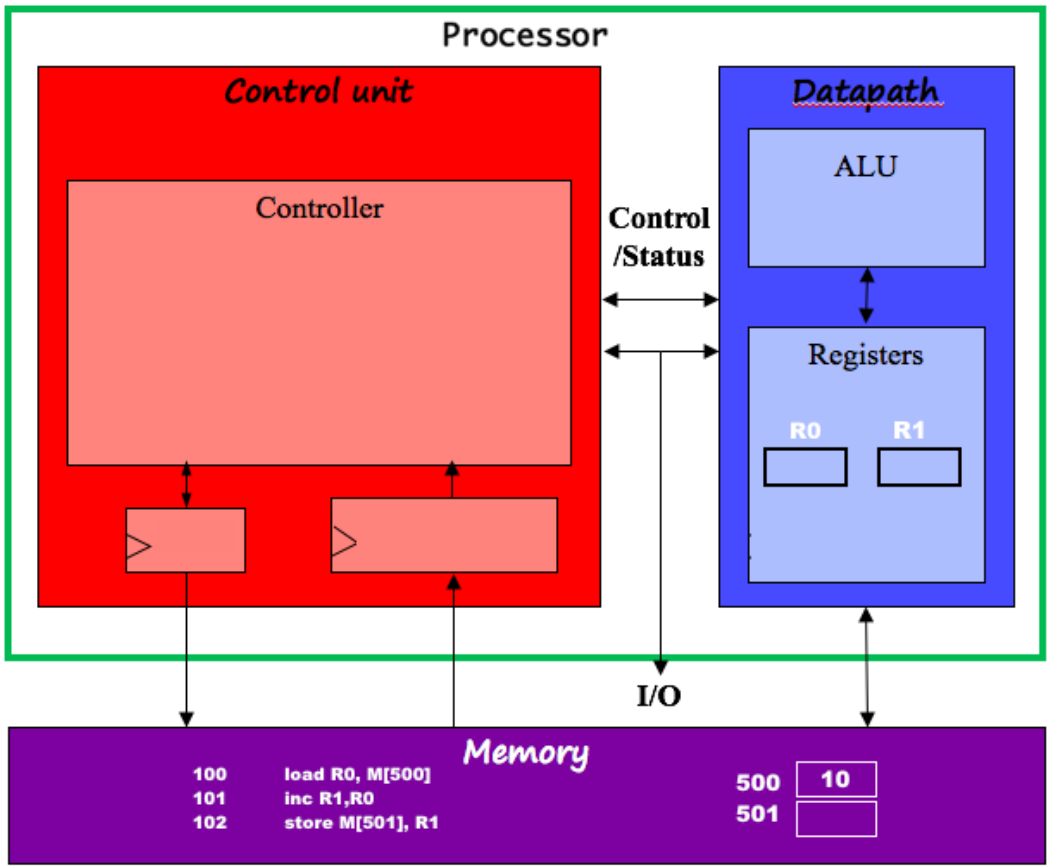
- Datapath Operations
  - Load : Read memory location into register
  - ALU operation: Input certain registers through ALU, store back in register
  - Store : Write register to memory location
- Control Unit
  - Control unit: configures the datapath operations
    - Sequence of desired operations ("instructions") stored in memory "program"
  - Instruction cycle Äì broken into several sub-operations, each one clock cycle, e.g.:
    - Fetch: Get next instruction into IR
    - Decode: Determine what the instruction means
    - Fetch operands: Move data from memory to datapath register
    - Execute: Move data through the ALU
    - Store results: Write data from register to memory

Instruction Cycles

```
0:   INPUT S0,(S1);
                ;Fetch :  get next instruction into IR
                ;Decode:  Determine what the instruction means
                ;Fetch operands:  Move data from memory to datapath register
                ;Execute:  Move data through the ALU
                ;Store results:  Write data from register to memory
1:   ADD S0, 01;
                Fetch :  get next instruction into IR
                Decode:  Determine what the instruction means
                Fetch operands:  Move data from memory to datapath register
                Execute :Move data through the ALU
                Store results:  Write data from register to memory
2:   STORE S0,(S1)
                Fetch :  get next instruction into IR
                Decode:  Determine what the instruction means
                Fetch operands:  Move data from memory to datapath register
                Execute :Move data through the ALU
                Store results:  Write data from register to memory
```

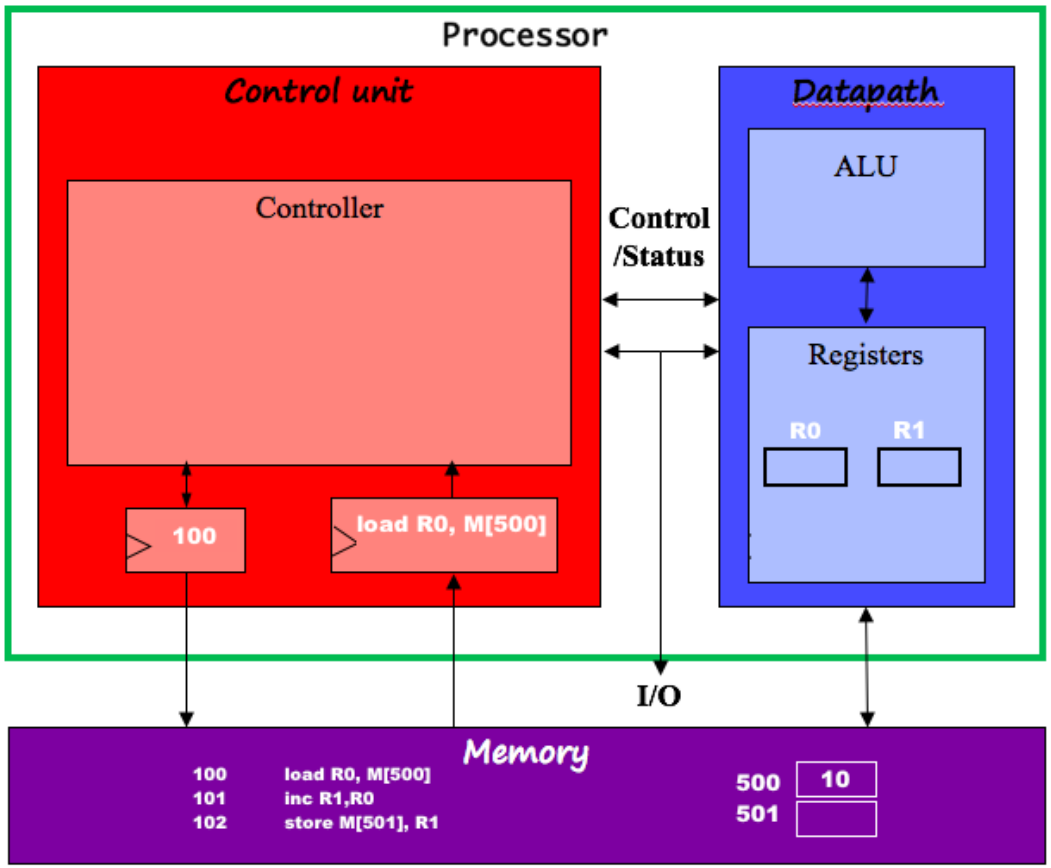
General-Purpose Processors

Instruction Cycle:

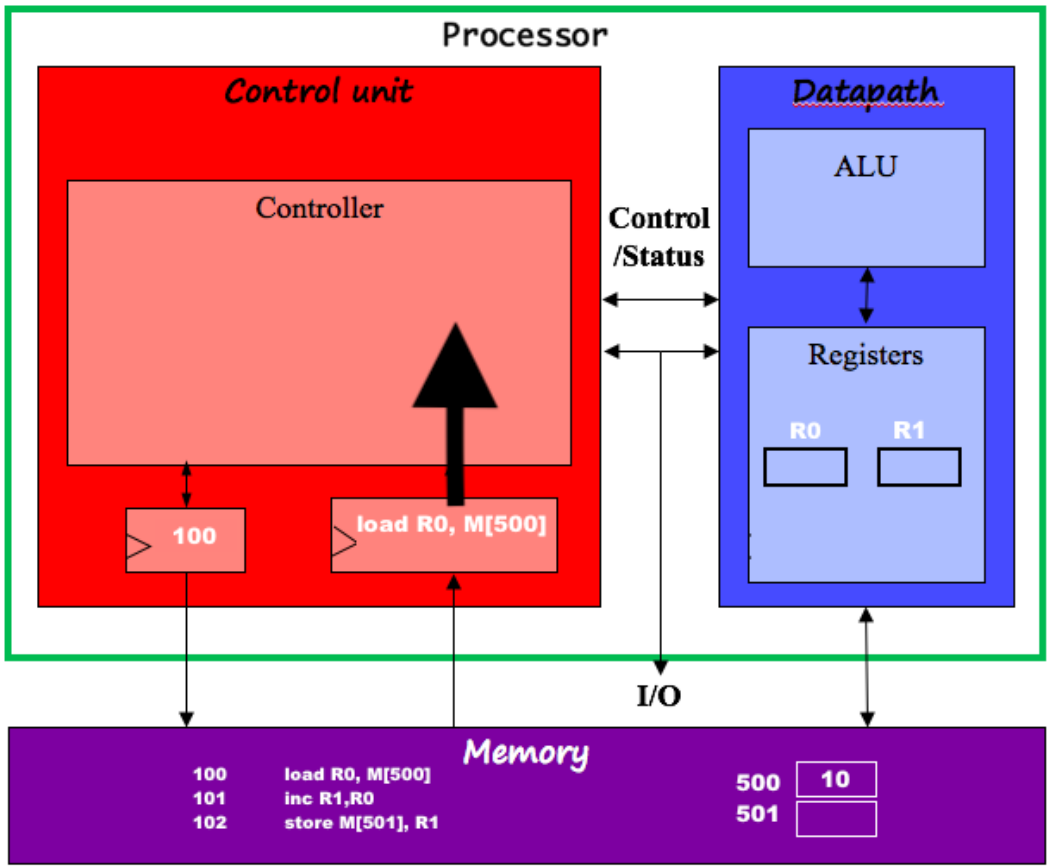


General-Purpose Processors

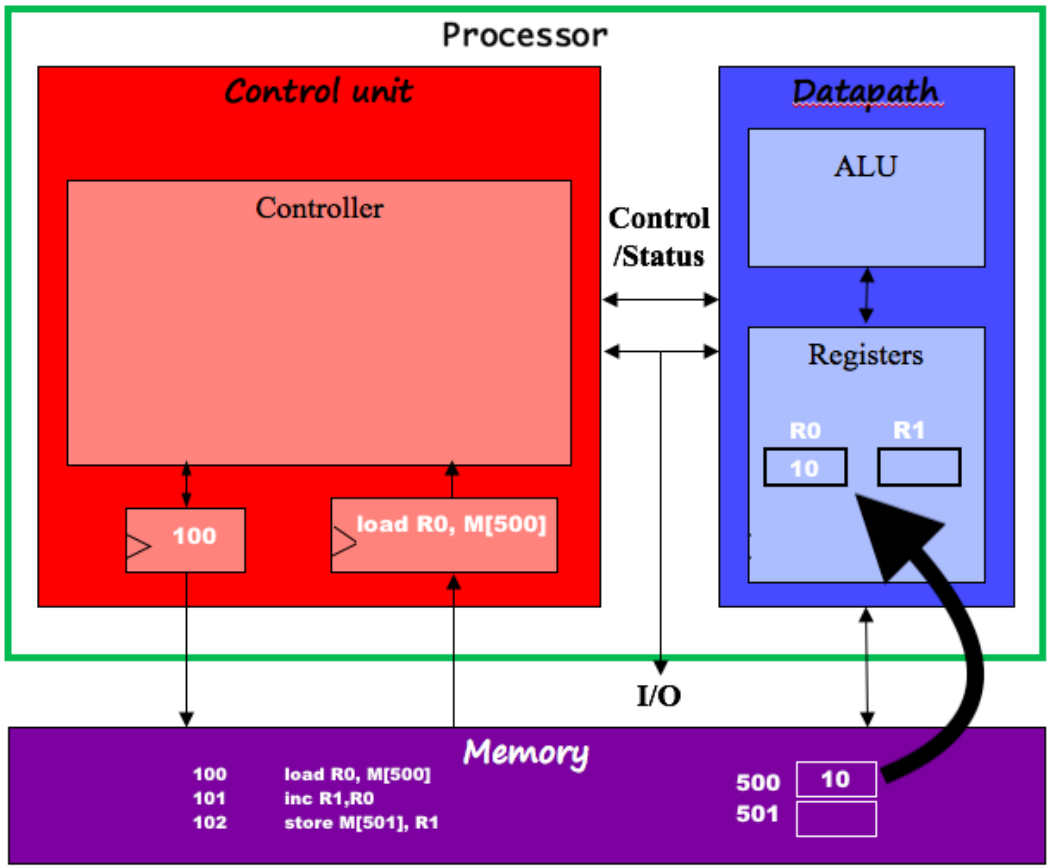
Instruction Cycle: Fetch



Instruction Cycle: Decode

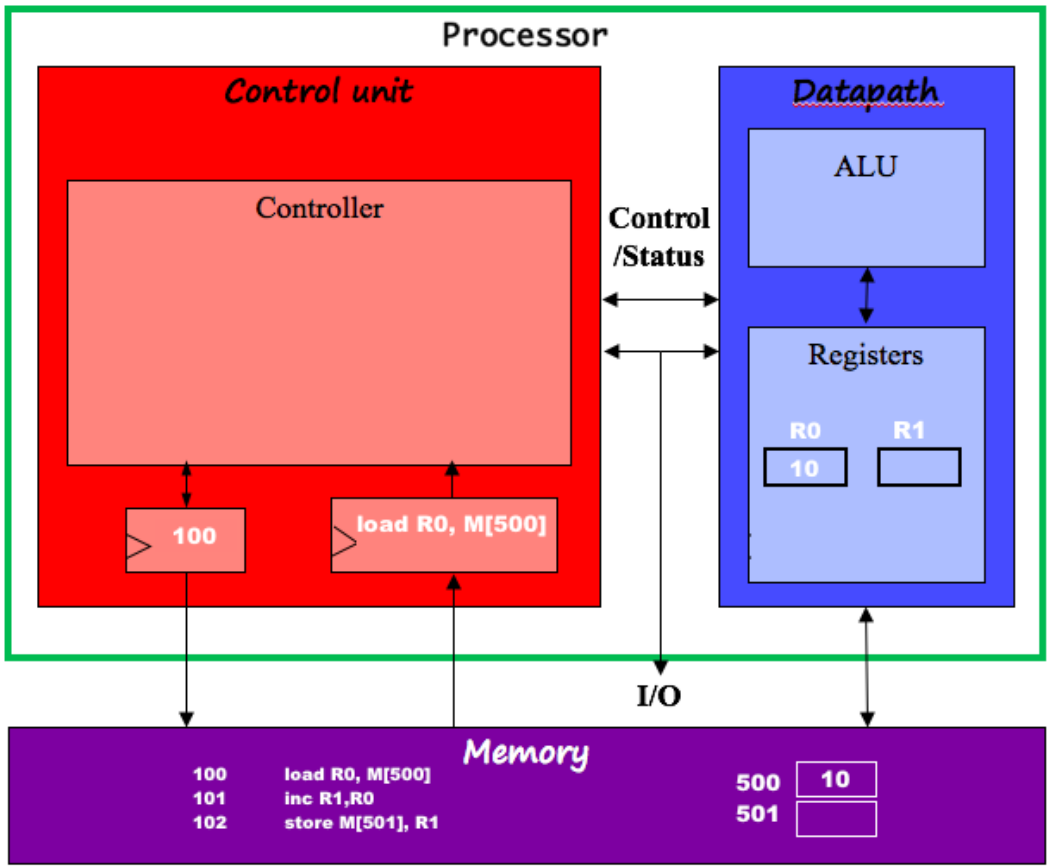


Instruction Cycle: Fetch operand



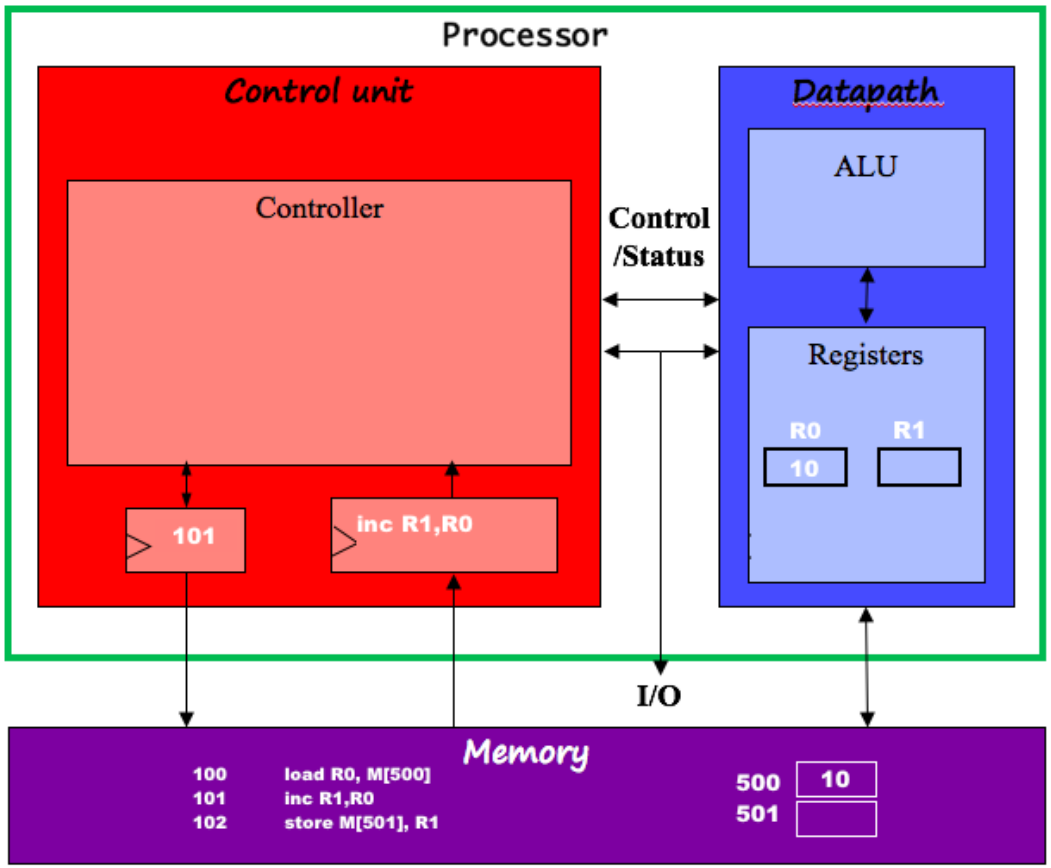
General-Purpose Processors

Instruction Cycle: Execute & Store results

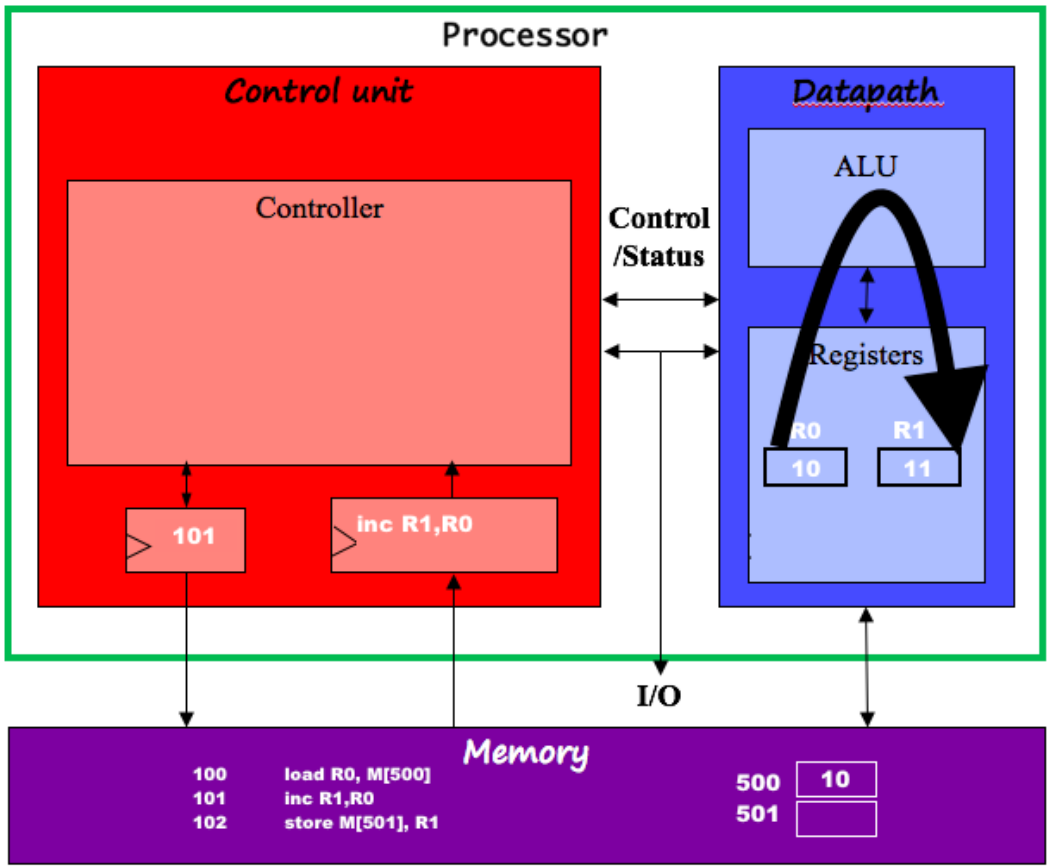


General-Purpose Processors

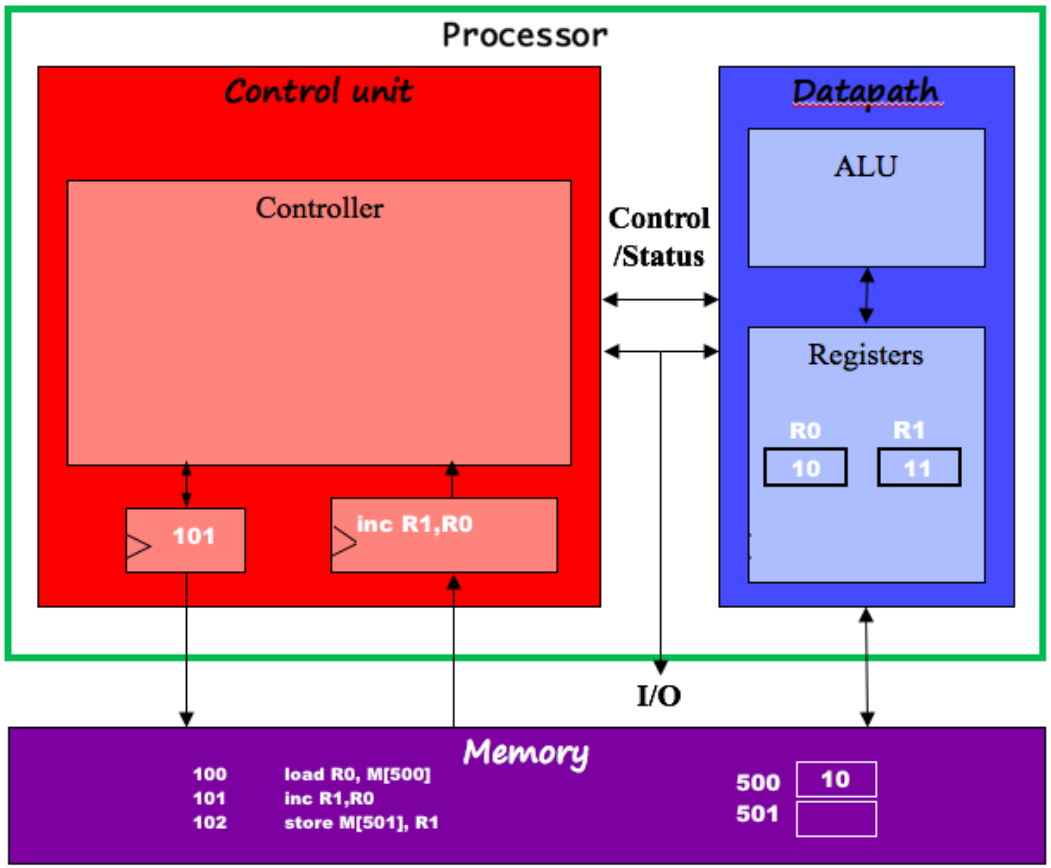
Instruction Cycle: Fetch & Decode & Fetch Operand



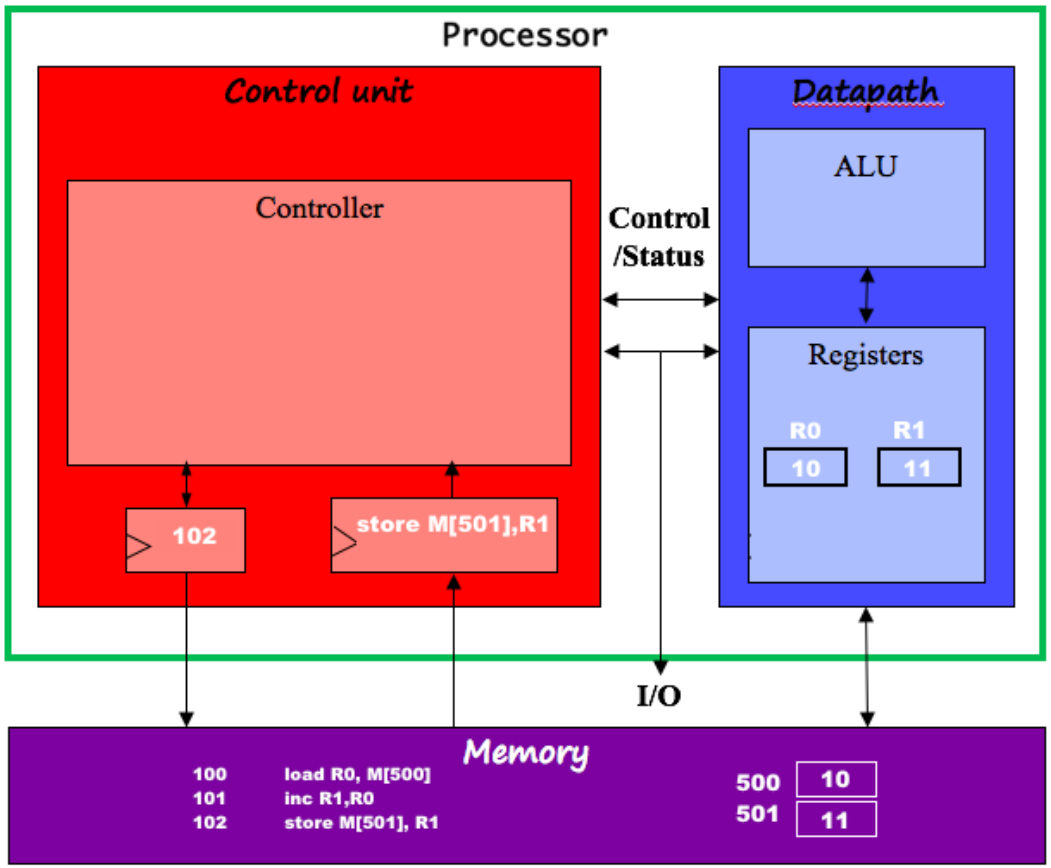
Instruction Cycle: Execute



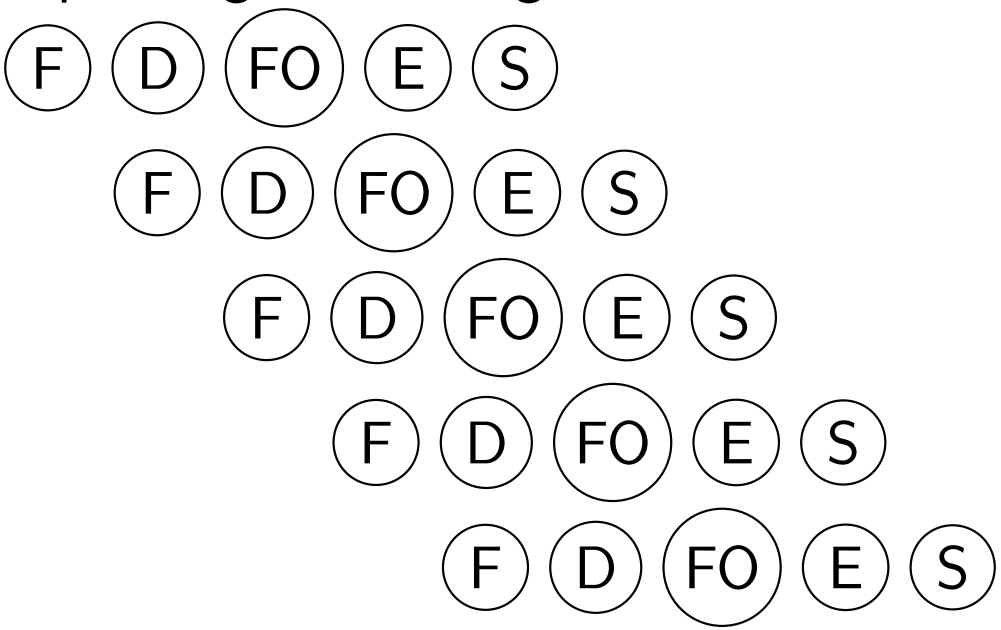
Instruction Cycle: Store Result



Instruction Cycle: Store Result



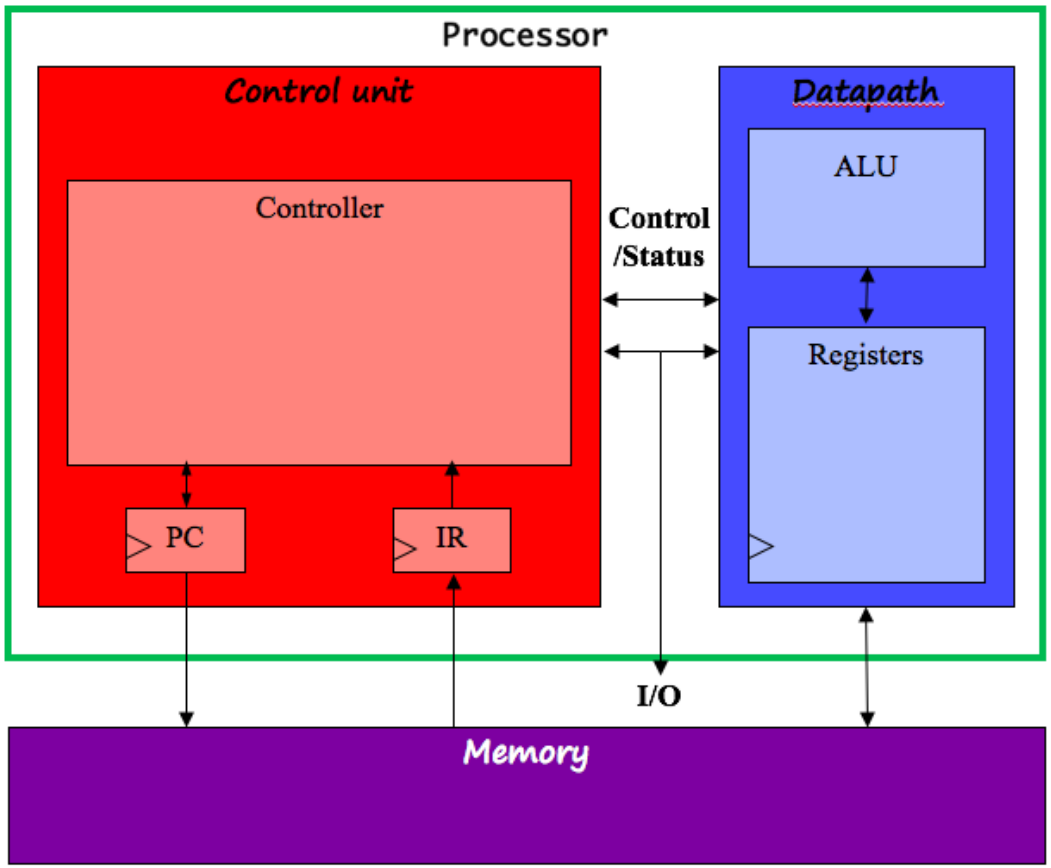
Pipelining: Increasing Instruction Throughput



General-Purpose Processors

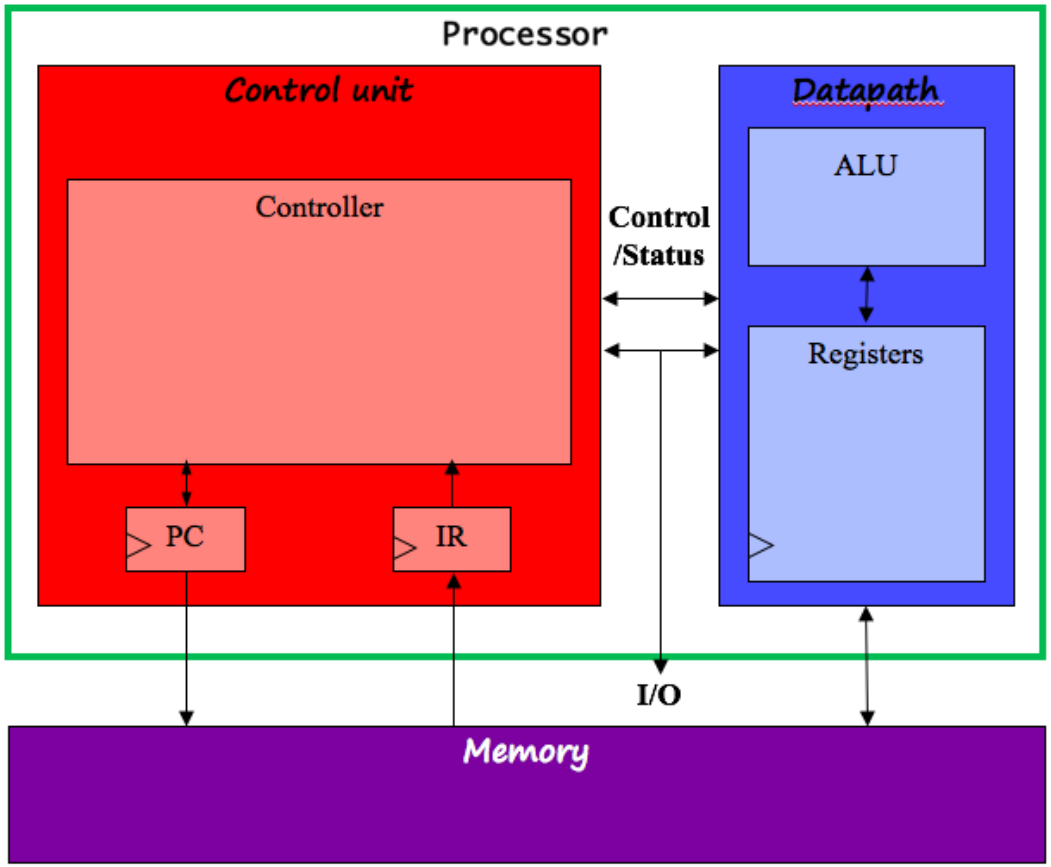
Performance can be improved by:

- Faster clock (but there is a limit)
- Pipelining: slice up instruction into stages, overlap stages
- Multiple ALUs to support more than one instruction stream



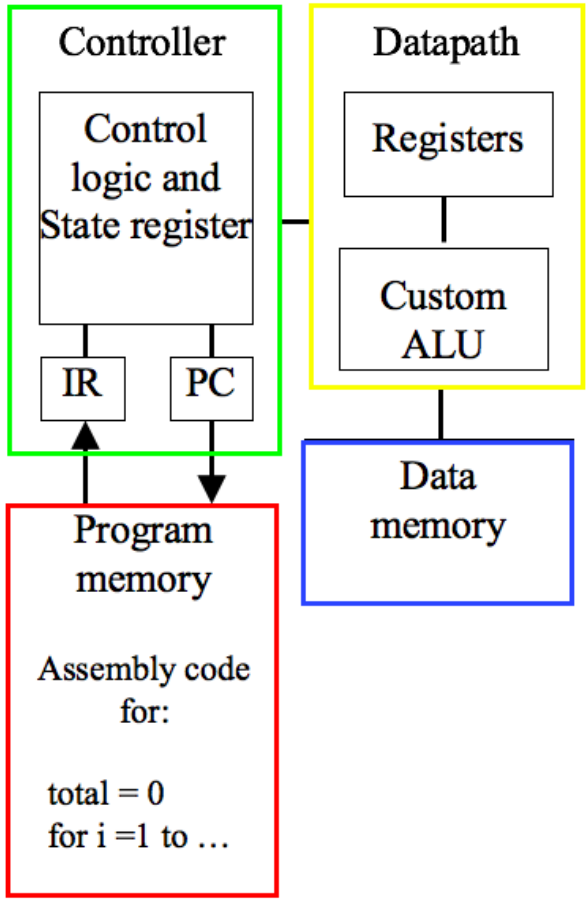
Architectural Considerations

- Faster clock (but thereÄôs a limit)
- Pipelining: slice up instruction into stages, overlap stages
- Multiple ALUs to support more than one instruction stream



Application-specific processors

- Programmable processor optimized for a particular class of applications having common characteristics
- Features
  - Program memory
  - Optimized datapath
  - Special functional units
- Benefits
  - Some flexibility
  - good performance
  - size and power



Application-specific processors: Intel 8051

8051 Architecture Advantages

- Fast I/O operations and fast access to on-chip RAM in data space
- Efficient and flexible interrupt system
- Low-power operation

8-bit (classic and extended 8051) devices include an efficient interrupt system designed for real-time performance and are found in more than 65% of all 8-bit applications. ▸ What is the 8051 doing in the year 2008 ?

Over 1000 variants are available ▸ Keil:8051, with peripherals that include analog I/O, timer/counters, PWM, serial interfaces like UART, 2 I C, LIN, SPI, USB, CAN, and on-chip RF transmitter supporting low-power wireless applications. Some architecture extensions provide up to 16MB memory with an enriched 16/32-bit instruction set.

Applications:

Bilgi Güvenliği ve Akıllı Kartlar, ▸ II. Ağ ve Bilgi Güvenliği Ulusal Sempozyumu, EMO, 2008

Smart Card Interface Design Example ▸ Microsemi

Programmer's View

A programmer writes the program instructions that carry out the desired functionality on the general-purpose processor.

- Programmer doesn't need detailed understanding of architecture
  Instead, needs to know what instructions can be executed
- Two levels of instructions:
  - Assembly level (lying between machine language and high level language)
  - Structured languages (C, C++, Java, etc.)
- Most development today done using structured languages
  - But, some assembly level programming may still be necessary
  - Drivers: portion of program that communicates with and/or controls (drives) another device
    - Often have detailed timing considerations, extensive bit manipulation
    - Assembly level may be best for these

Programmer's View: Instruction set

The assembly-language programmer must know the processors's instruction set. ▸ Keil:8051 Instruction Set Manual

An instruction : an opcode field and operand fields.

```
[opcode (specifies the operation)] [operand1 (loc.  of data)] [operand2 (loc.  of data)]
```

A machine language is a series of binary bytes representing instructions.
`ADD A, R7 // An 8051 instruction` ▸ binary codes of 8051
Instruction: 00101111b = `[00101] [111]= [[opcode] [register]]` ▸ Opcode of ADD

Programmer's View: Instruction set

The assembly-language programmer must know the processors's instruction set.
An instruction : an opcode field and operand fields.

```
[opcode (specifies the operation)] [operand1 (loc.  of data)] [operand2 (loc.  of data)]
```

- Data-transfer instructions
- Arithmetic/logical instructions
- Branch instructions
  - Unconditional Jump
  - Conditional Jump
  - Return instruction
  - Call instruction (saves the add. of current inst.)

Source operands serve as input to the operation, while a destination operand stores the output.

Programmer's View: Instruction set

- Data-transfer instructions: ▸ 80C51 instruction set (page 90)

```
MOV <destination>, <source> // An 8051 instruction
XCH A, <source> // the acc.  and the addressed byte to exch.
data
```

▸ Keil: 80C51 Ins. Set Manuel

- Arithmetic/logical instructions ▸ 80C51 instruction set (page 90)

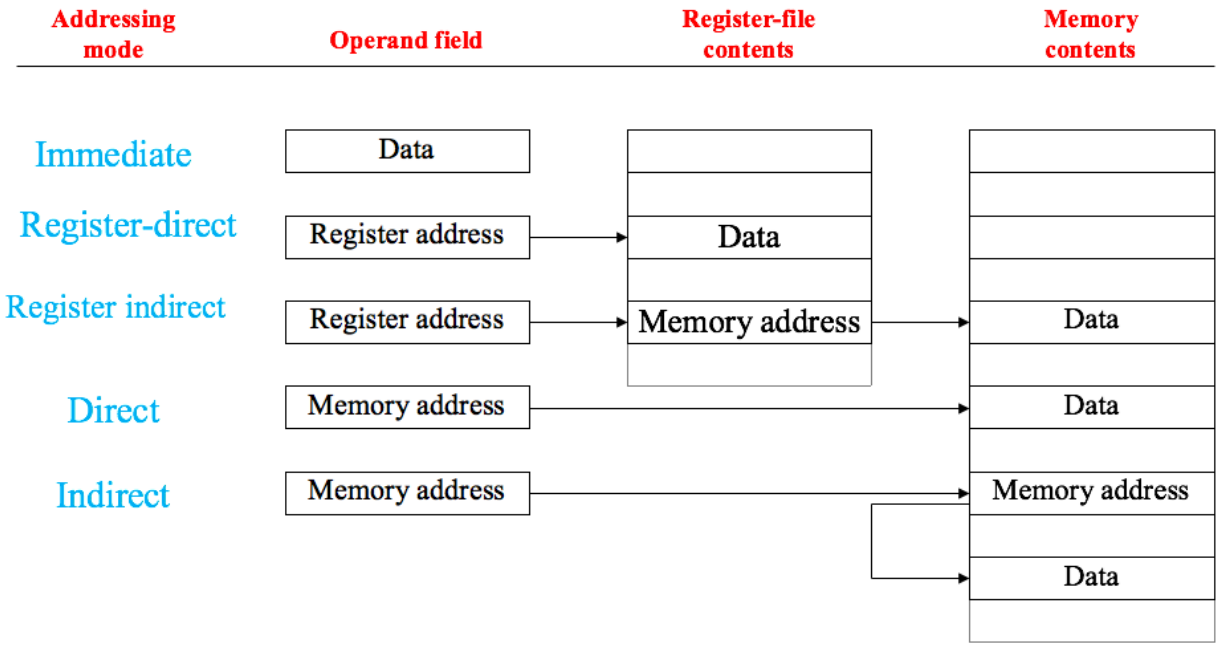
```
ADD A, 7Fh // An 8051 instruction
INC 7Fh
ANL A, #01010011b // AND operation
RRC A // Rotate ints.
```

- Branch instructions ▸ 80C51 instruction set (page 93)
  - Unconditional Jump `AJMP addr11`
  - Conditional Jump `JNZ rel`
  - Return instruction `RET`
  - Call instruction (saves the add. of current inst.)`ACALL addr11`

Source operands serve as input to the operation, while a destination operand stores the output.

Programmer's View: Instruction set

The operand field may indicate the data's location through one of several addressing modes :



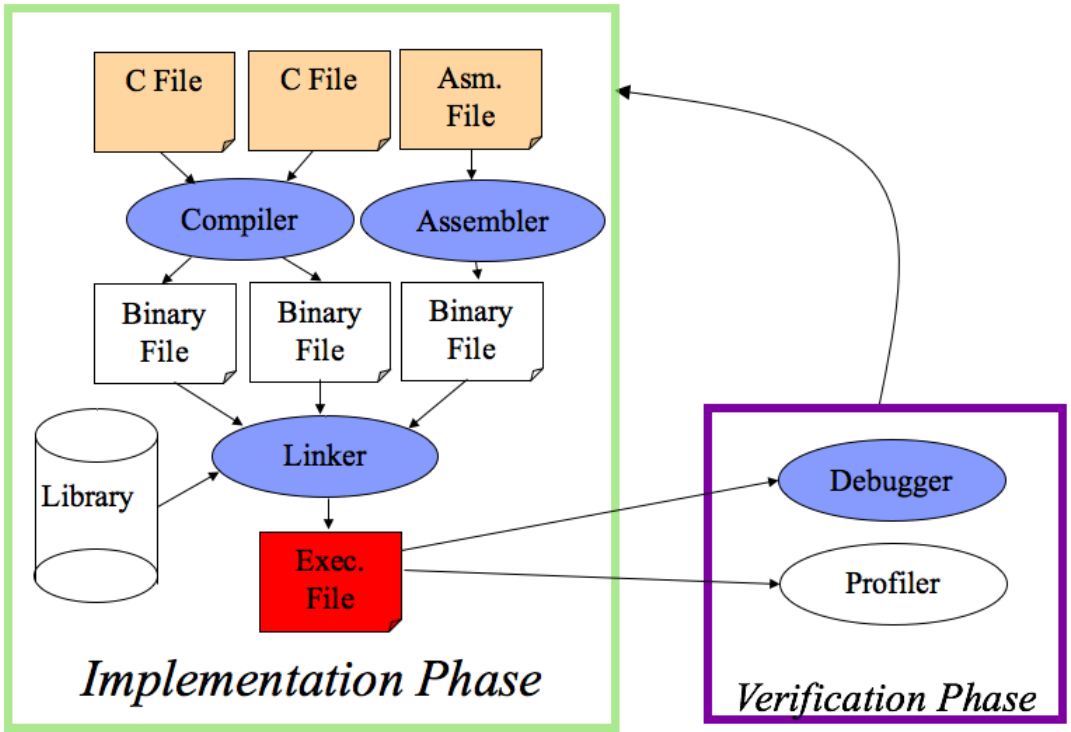

```
ADD A, #55H // ACC + 55H → ACC
ADD A, R5 // ACC + R5 → ACC
ADD A,@R0
ADD A, direct
MOVX @DPTR
```

The embedded systems programmer must be aware of

- Program and data memory space (▸ Figures 1 and 2)
- Registers (for data-transfer instruction etc.): How many are there?
- Input and output (I/O) facilities (to help communicate with other devices)
- Interrupts :
  An interrupt causes the processor to suspend execution of the main program, and instead jump to an Interrupt Service Routine (ISR) that fulfills a special, short-term processing need.

Development Environment

- Development processor: on which we write and debug our programs
- Target processor : that the program will run on in our embedded system



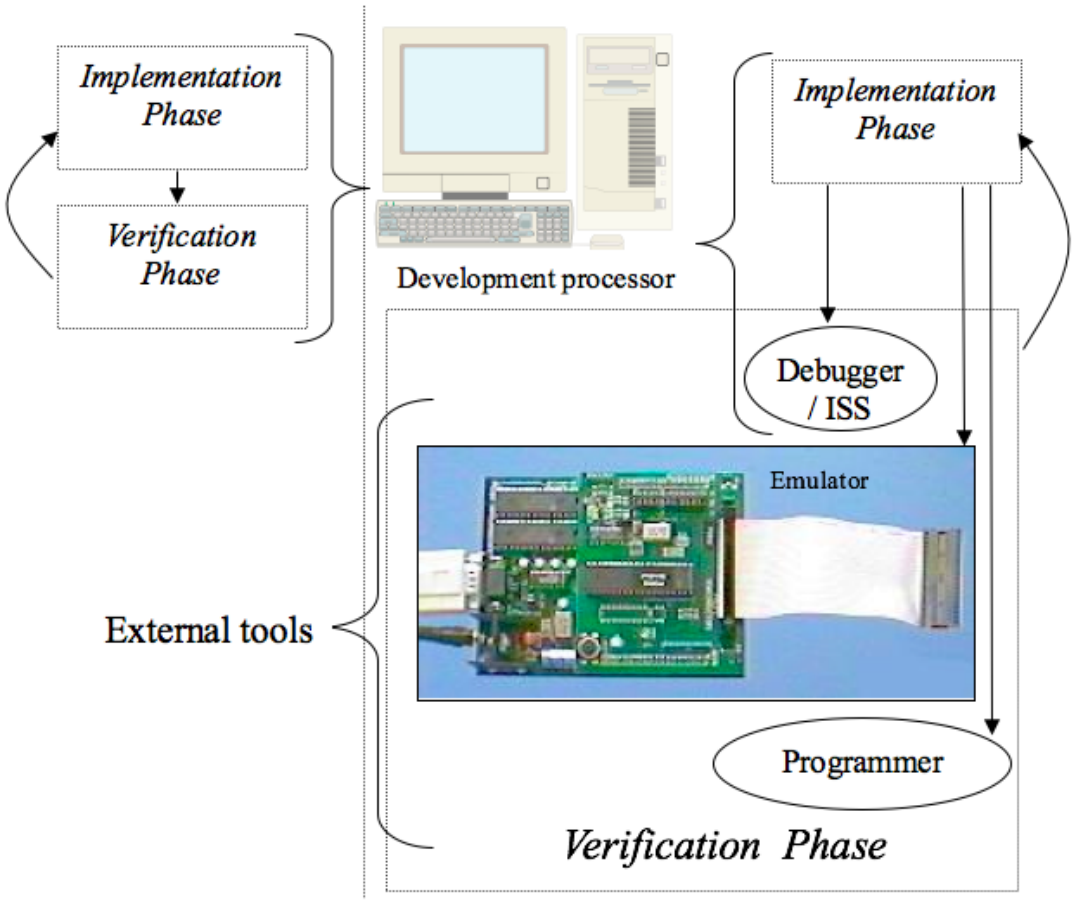
Development Environment

- Assemblers translate assembly instructions to binary machine instructions ▸ Keil: A51 Macro Assembler
- A linker allows a programmer to create a program in separately-assembled files ▸ Keil: BL51 Code Banking Linker/Locator.
- Compilers translate structured programs into machine (or assembly) programs ▸ Keil: C51 ANSI C Compiler.
- A cross-compiler executes on one processor (our development processor), but generates code for a different processor (our target processor).
- Debuggers help programmers evaluate and correct their programs ▸ Keil: FlashMON51 Target Monitor
- Emulators support debugging of the program while it executes on the target processor.

All this tool in integrated development environment (IDE)

▸ Keil: Professional Developer's Kit

Development Environment



- Instruction set simulator (ISS):
  - Gives us control over time Äì set breakpoints, look at register values, set values, step-by-step execution, ...
  - But, doesnÄôt interact with real environment
- Download to board
  - Use device programmer
  - Runs in real environment, but not controllable
- Compromise: emulator
  - Runs in real environment, at speed or near
  - Supports some controllability from the PC

▸ Keil: Evaluation Boards and Emulators for 8051