CPE 390: Microprocessor Systems

Spring 2018

Lecture 3

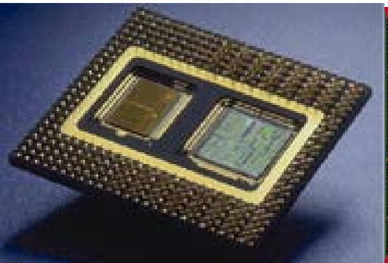
Elements of a Microcomputer System

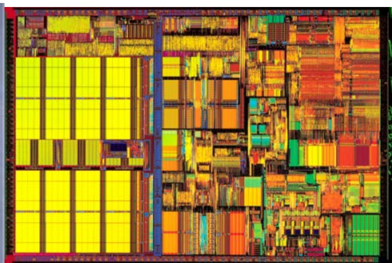
Bryan Ackland

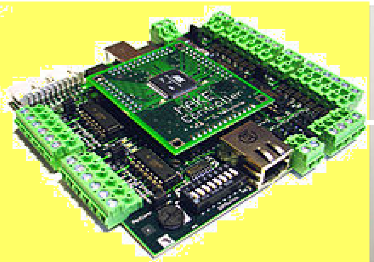
Department of Electrical and Computer Engineering

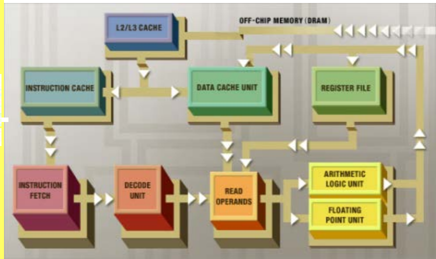
Stevens Institute of Technology

Hoboken, NJ 07030

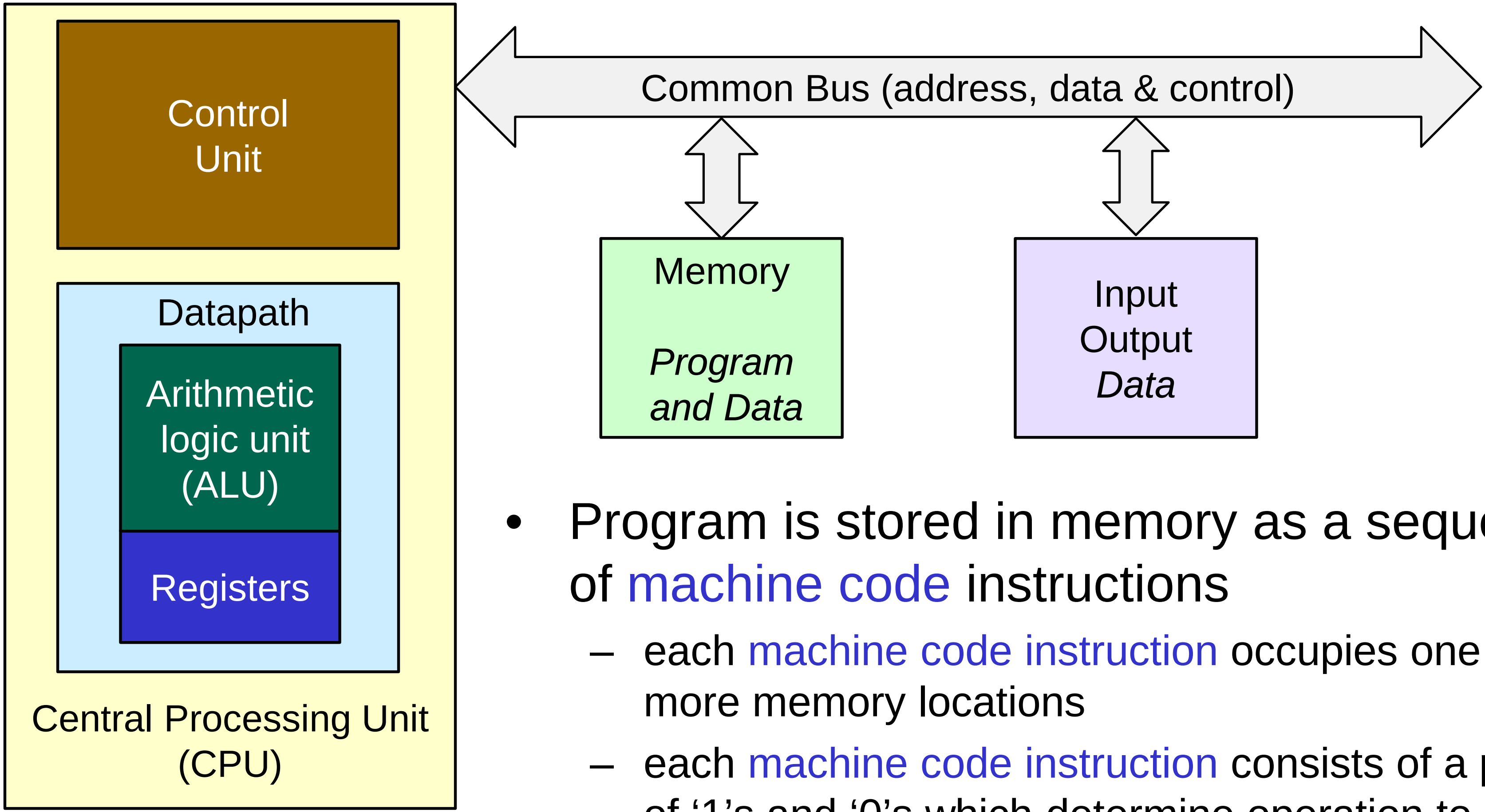








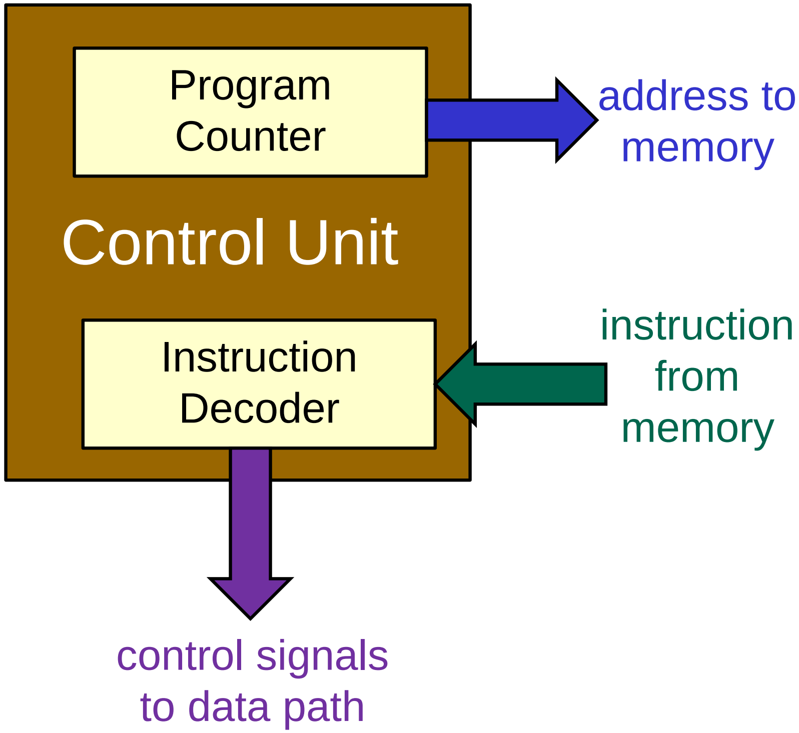
Stored Program Digital Computer



Control Unit
Datapath
Arithmetic logic unit (ALU)
Registers
Central Processing Unit (CPU)
Common Bus (address, data & control)
Memory
Program and Data
Input Output
Data

- Program is stored in memory as a sequence of machine code instructions
  - each machine code instruction occupies one or more memory locations
  - each machine code instruction consists of a pattern of ‘1’s and ‘0’s which determine operation to be performed by the CPU
  - mapping of machine code instructions to CPU operations is sometimes called instruction set architecture

CPU Control Unit



- Contains special register known as program counter
  - contains memory address of next instruction to be executed

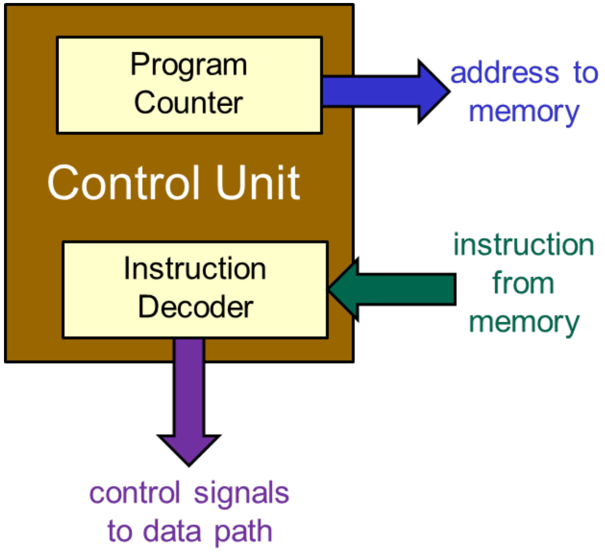
- Control unit fetches from memory the machine code instruction whose address is given by the program counter

- Control unit decodes the instruction

- Control unit executes the instruction

Instruction Execution

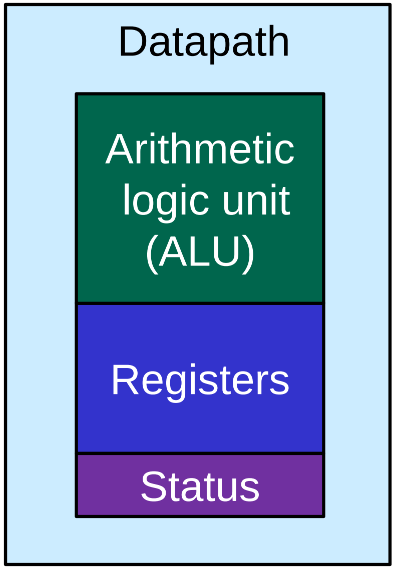
- Control unit executes the instruction including:
  - fetching required data operands from registers and/or memory
  - controlling ALU to perform any required data operation
  - storing any results in registers and/or memory
  - updating program counter to point to address of next instruction to be executed



- Program instructions are normally stored in sequential locations in memory
- Control unit recognizes (conditional) branch instructions which alter the normal program flow

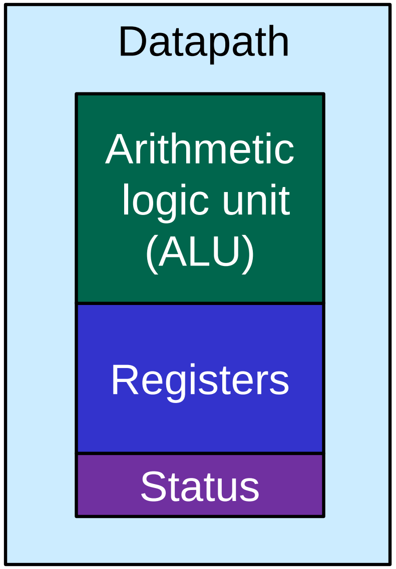
CPU Datapath - Registers

- Temporary storage location inside CPU
- Hold data or address of data to be processed
- Provide operands to ALU and receive results from ALU
- number of registers can range from:
  - 1 (simple microcontroller with accumulator) to
  - over 100 (128 in Intel Itanium)



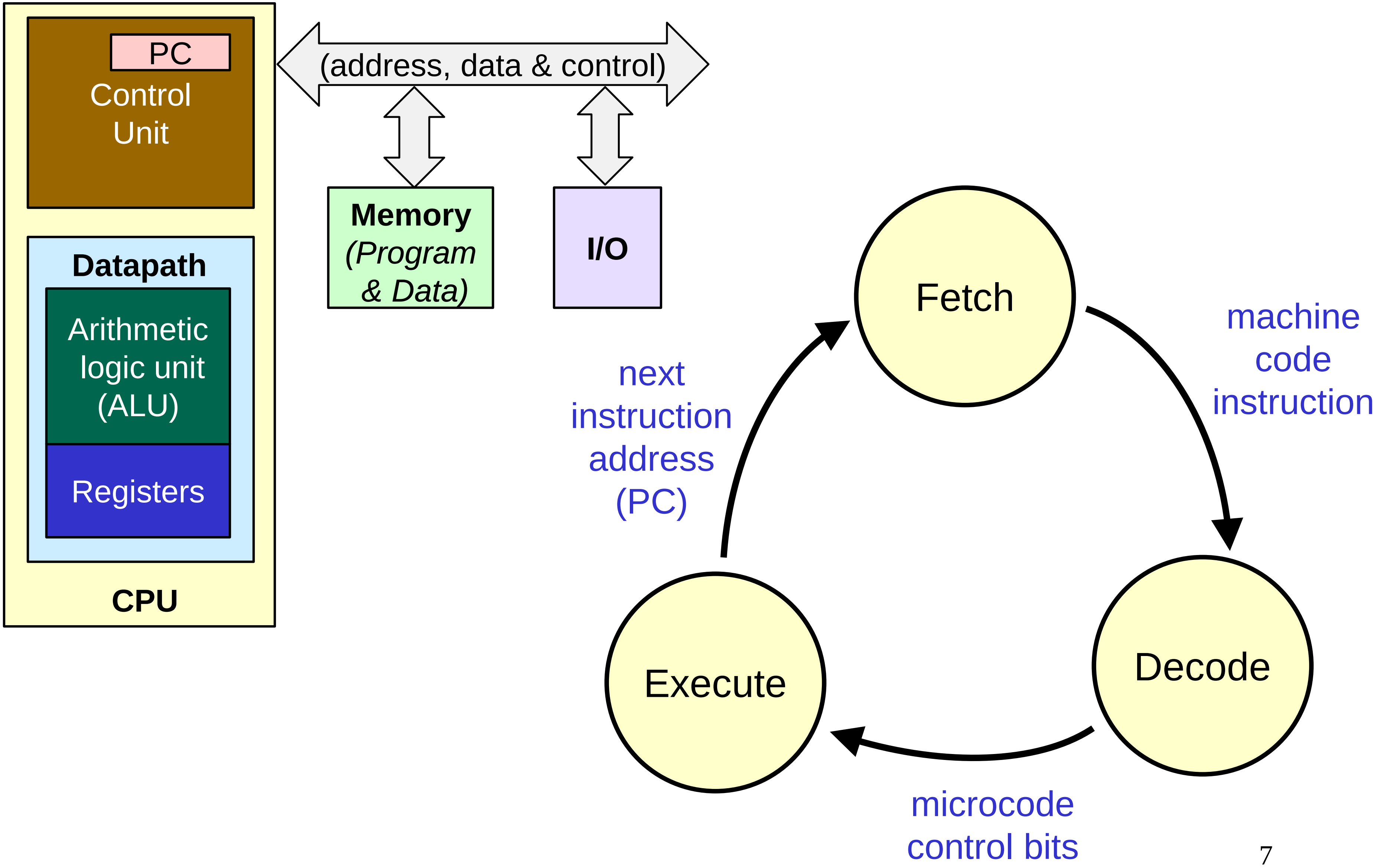
- General purpose vs. special purpose registers
- Status (condition code) register holds bits (flags) that reflect result of last ALU operation (e.g. carry, zero, sign)
  - used by control unit in branching operations
- Faster access than memory (fewer clock cycles)
- Simpler instruction format (limited address range)

CPU Datapath – Arithmetic & Logic Unit (ALU)

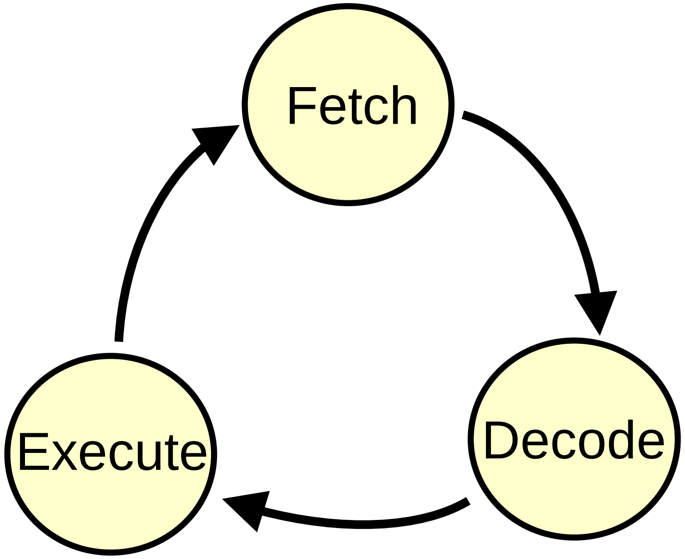


- Performs simple arithmetic and logic operations on data stored in registers and/or memory
- Add, subtract, and, or, shift, increment, multiply, etc.
- Operation depends on current machine code instruction
- Simple operations (register to register add) may execute in one clock cycle
- May include special purpose complex operations
  - e.g. square-root, floating point, divide, complex number arithmetic
  - typically take multiple clock cycles
- Results stored in registers and/or memory
- Status register updated according to result

Basic Instruction Cycle



Basic Instruction Cycle



- **Fetch:**
  - Read machine code instruction from memory at address supplied by program counter (PC)
  - May take multiple memory cycles if instruction occupies more than one memory location
- **Decode:**
  - Determine operations to be performed and generate microcode bits to control CPU hardware
- **Execute:**
  - Fetch required operands from memory and/or registers
  - Perform required ALU operations
  - Store results in registers and/or memory
  - Update PC

Operations Performed by Instructions

- Range of operations performed by individual instructions varies according to instruction set architecture and is different for each microprocessor family

- Microprocessor instructions typically:
  - move data between registers and memory
  - perform arithmetic and logic operations on data stored in registers and/or memory
  - update PC to implement branching (conditional and unconditional)
  - input data *from* and output data *to* external peripherals
  - set internal registers in CPU to control operation of specific functionality (e.g. interrupts, timers, I/O)

Instruction Examples (HCS12)

- Memory is byte addressed
  - each memory location contains 8-bits of data

- One-byte instruction:

address *n*: 0 1 0 1 0 0 0 1

complement all bits in accumulator (register) B

- Two-byte instruction:

address *n*: 1 0 0 1 1 0 1 1

add contents of memory location whose address is in next instruction byte to accumulator (register) A

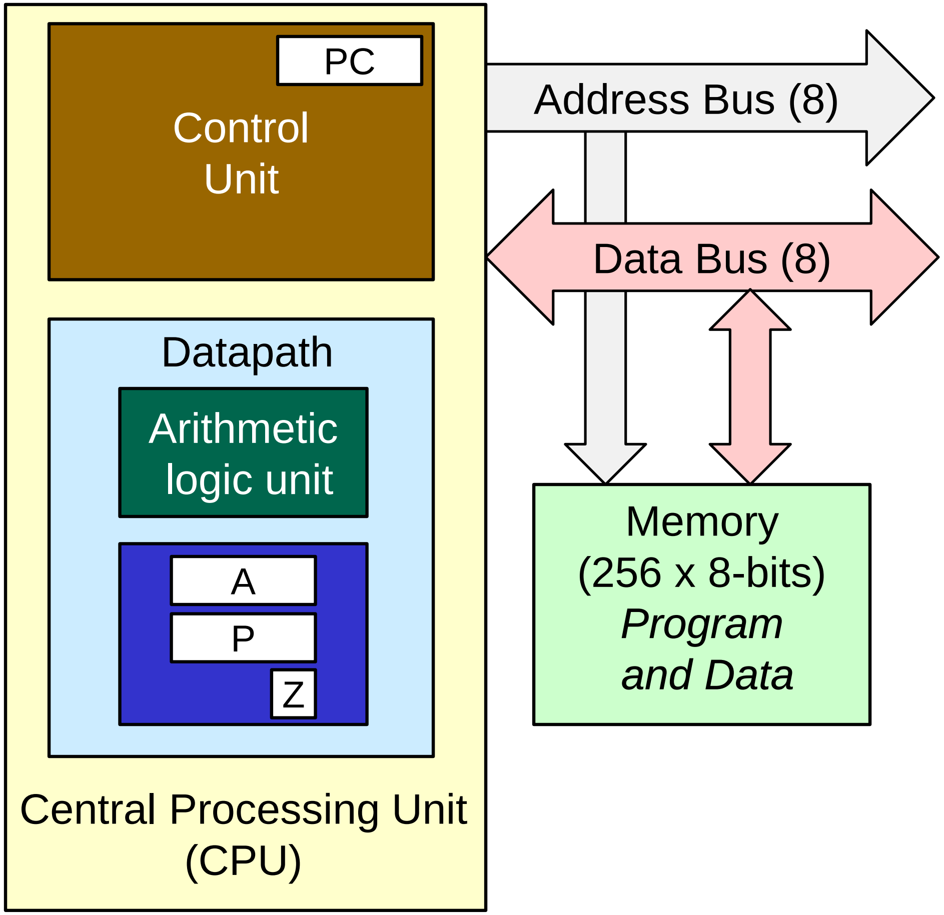
address *n+1*: 0 0 0 0 0 0 1 1

data is in memory address 3

i.e., add contents of memory address 3 to accumulator A

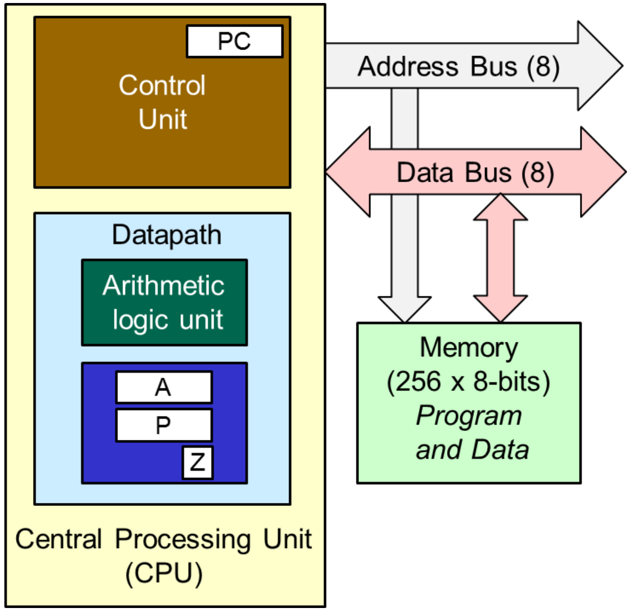
Program Execution Example

- Suppose we have very simple processor:
  - 8-bit memory address and 8-bit data bus
    - up to 256 x 8-bit words of memory
  - CPU has 8-bit ALU and 3 registers:
    - 8-bit program counter (PC)
    - 8-bit accumulator (A)
    - 8-bit pointer register (P)
    - 1-bit zero flag to indicate last arithmetic result was zero



Simple Processor Instruction Set

- Suppose our simple processor understands only 7 different instructions:



Instruction	Hexadecimal Machine Code	Meaning
lda #val	27 xx	Load 8-bit value into A
ldp #val	E7 xx	Load 8-bit value into P
sta addr	91 aa	Store A into memory location at addr.
adda @ptr	3B	Add data in mem. loc. pointed to by P to A
incp	62	Increment P
decm addr	1C aa	Decrement contents of mem. loc. at addr.
bnz offset	55 zz	If last result was not zero, add (signed) offset to PC

Sample Program (Example)

- Use simple processor to add contents of memory locations 20-23 (hex) and store result in location 24 (hex)

```
            lda     #4       ; load A with number 4
            sta     0x30     ; and store in mem. loc. 30 (hex) as counter
            ldp     #0x20    ; set P to point at mem. loc. 20 (hex)
            lda     #0       ; load A with zero
   loop:    adda    @ptr     ; add data pointed to by P to acc.  A
            incp             ; increment data pointer
            decm    0x30     ; decrement counter
            bnz     loop     ; if not done, go to loop
            sta     0x24     ; store result in mem. loc. 24 (hex)
```

Machine Code (Example)

Machine Code loaded into Memory:

Assembly	Address	Machine Code
lda #4	0x00	27 04
sta 0x30	0x02	91 30
ldp #0x20	0x04	E7 20
lda #0	0x06	27 00
loop: adda @ptr	0x08	3B
incp	0x09	62
decm 0x30	0x0A	1C 30
bnz loop	0x0C	55 FA
sta 0x24	0x0E	91 24

ADDR:	
00	27
01	04
02	91
03	30
04	E7
05	20
06	27
07	00
08	3B
09	62
0A	1C
0B	30
0C	55
0D	FA
0E	91
0F	24

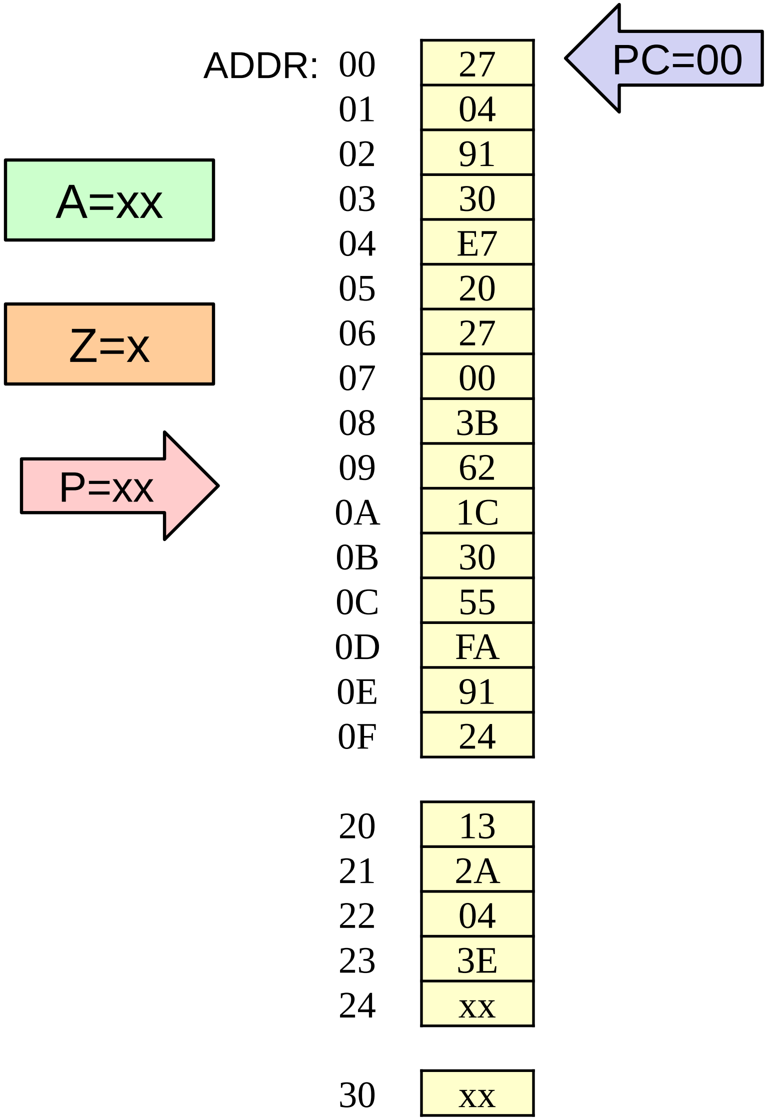
20	13
21	2A
22	04
23	3E
24	xx

30	xx

Suppose locations 20-23
preloaded with 13, 2A, 04 and 3E:

Initialization (Example)

1. Push reset button to begin program execution at 00
2. PC is loaded with 00
3. Other registers are not defined – filled with random junk
4. Begin program execution

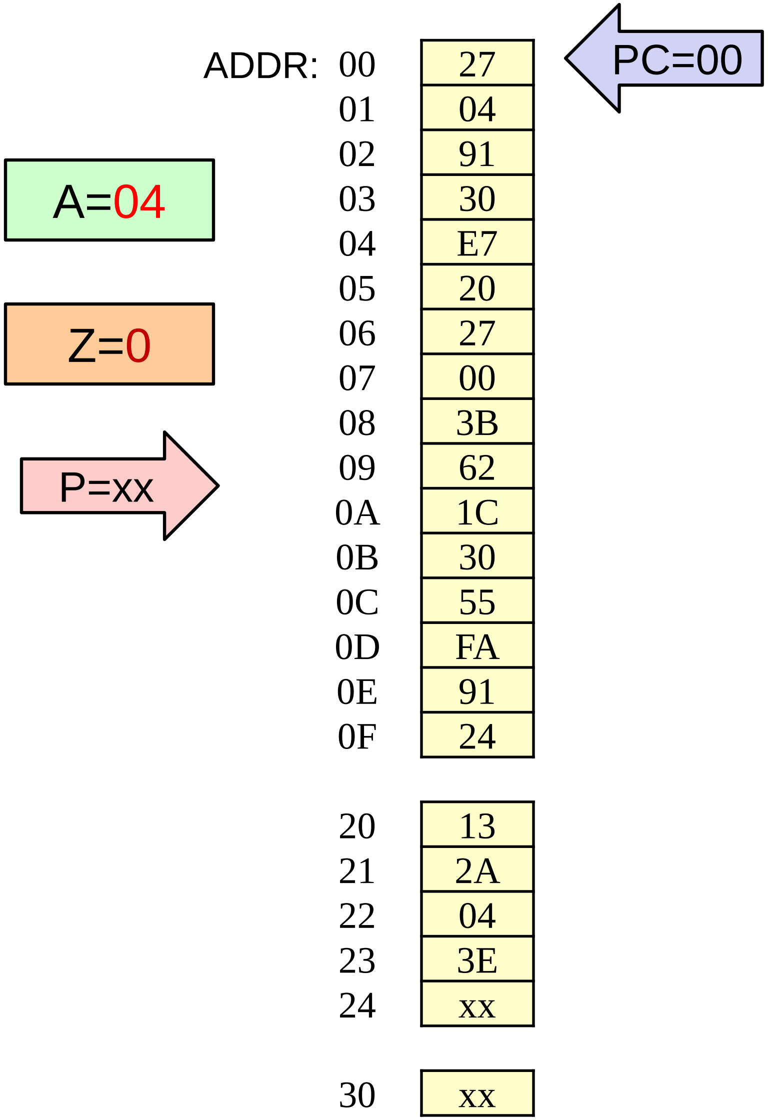


ADDR:	
00	27
01	04
02	91
03	30
04	E7
05	20
06	27
07	00
08	3B
09	62
0A	1C
0B	30
0C	55
0D	FA
0E	91
0F	24
20	13
21	2A
22	04
23	3E
24	xx
30	xx

Instruction 1 (Example)

lda #4

1. Fetch instruction from [00]
2. Decode instruction
3. Fetch data from [01]
4. Store data in A
5. Set Z flag according to data
6. Set PC to [02]

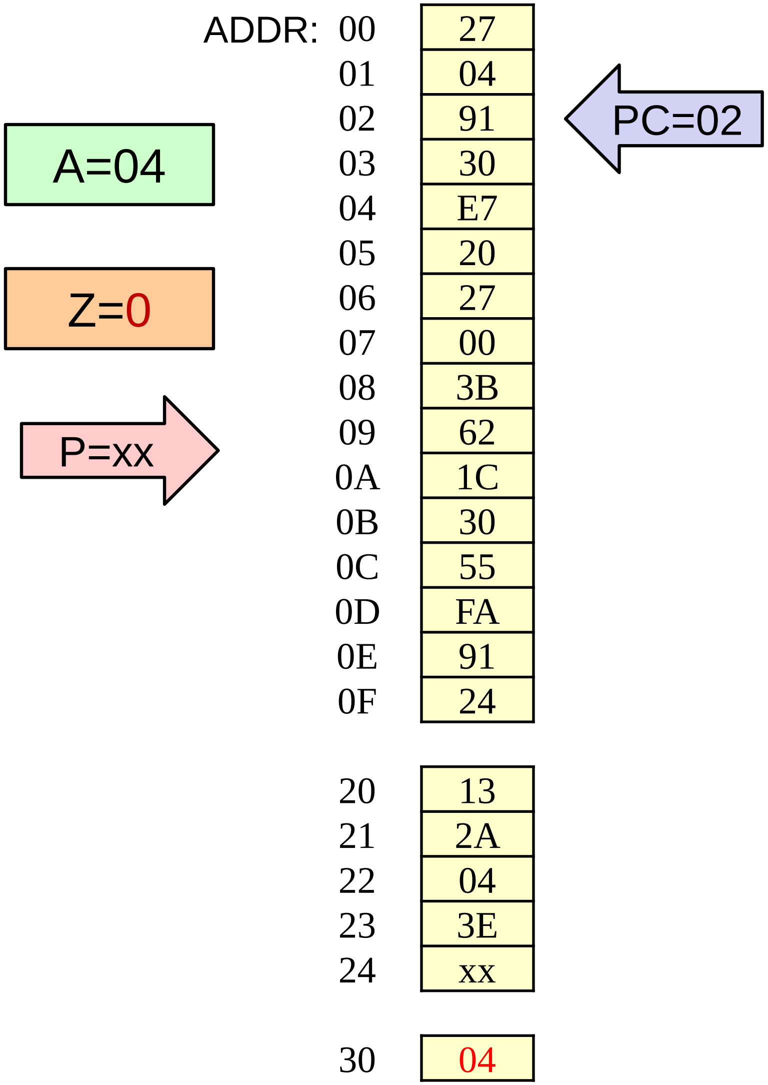


ADDR:	
00	27
01	04
02	91
03	30
04	E7
05	20
06	27
07	00
08	3B
09	62
0A	1C
0B	30
0C	55
0D	FA
0E	91
0F	24
20	13
21	2A
22	04
23	3E
24	xx
30	xx

Instruction 2 (Example)

sta 0x30

1. Fetch instruction from [02]
2. Decode instruction
3. Fetch address from [03]
4. Store [A] in mem. addr. 30
5. Set Z flag according to data
6. Set PC to [04]



A=04

Z=0

P=xx

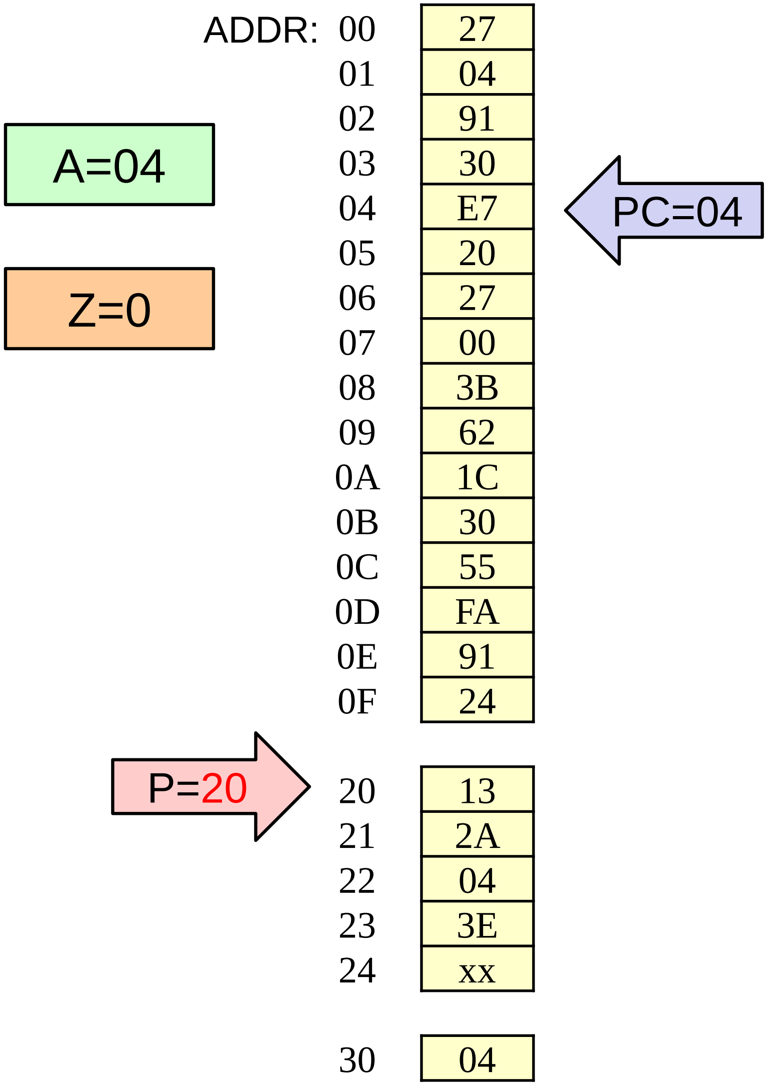
PC=02

ADDR:	
00	27
01	04
02	91
03	30
04	E7
05	20
06	27
07	00
08	3B
09	62
0A	1C
0B	30
0C	55
0D	FA
0E	91
0F	24
20	13
21	2A
22	04
23	3E
24	xx
30	04

Instruction 3 (Example)

ldp #0x20

1. Fetch instruction from [04]
2. Decode instruction
3. Fetch data from [05]
4. Store data in P
5. Set Z flag according to data
6. Set PC to [06]



A=04

Z=0

PC=04

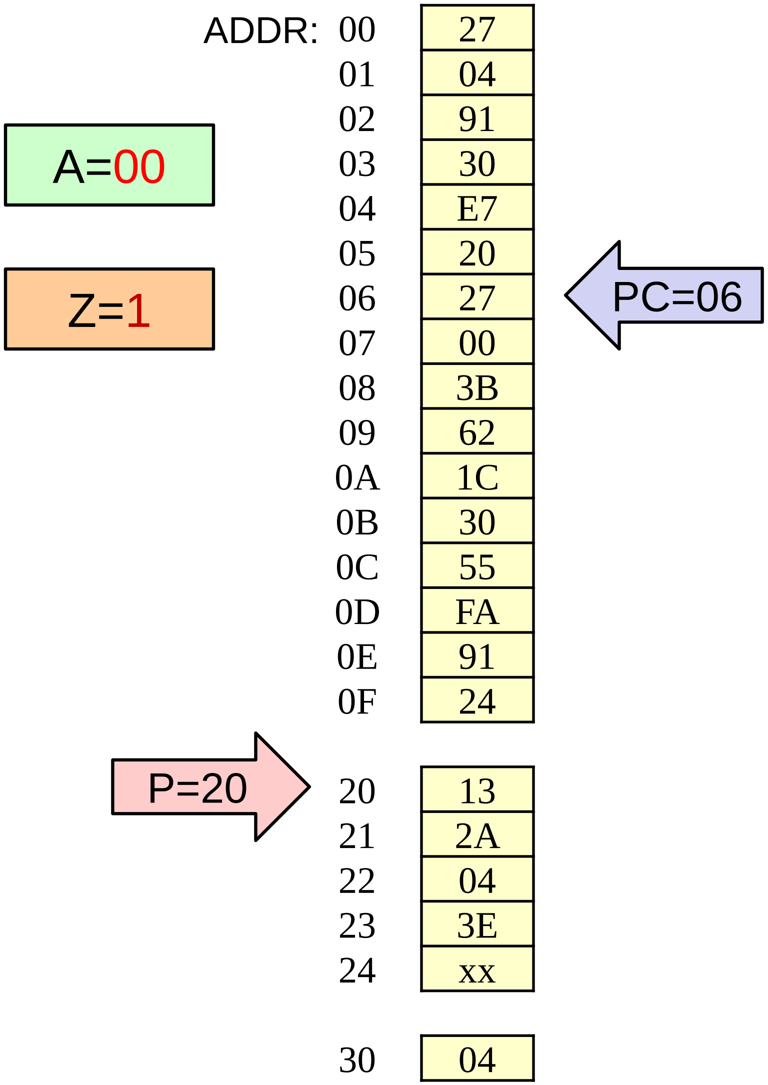
P=20

ADDR:	
00	27
01	04
02	91
03	30
04	E7
05	20
06	27
07	00
08	3B
09	62
0A	1C
0B	30
0C	55
0D	FA
0E	91
0F	24
20	13
21	2A
22	04
23	3E
24	xx
30	04

Instruction 4 (Example)

lda #0

1. Fetch instruction from [06]
2. Decode instruction
3. Fetch data from [07]
4. Store data in A
5. Set Z flag according to data
6. Set PC to [08]



A=00

Z=1

PC=06

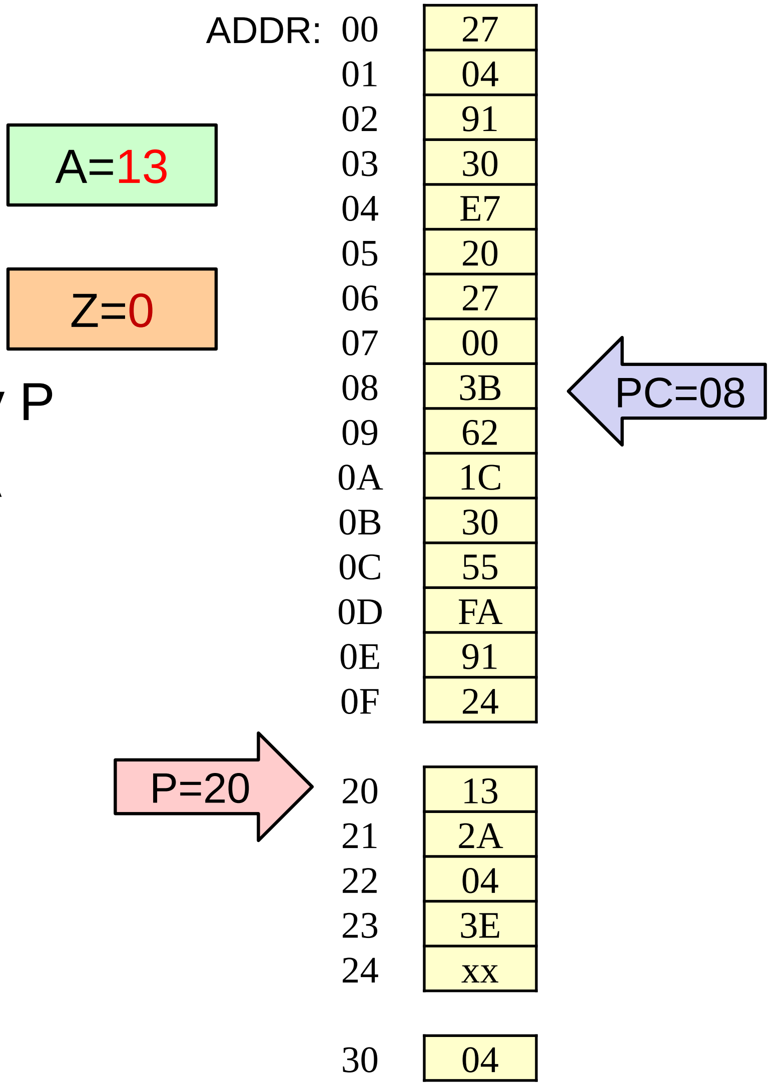
P=20

ADDR:	
00	27
01	04
02	91
03	30
04	E7
05	20
06	27
07	00
08	3B
09	62
0A	1C
0B	30
0C	55
0D	FA
0E	91
0F	24
20	13
21	2A
22	04
23	3E
24	xx
30	04

Instruction 5 (Example)

adda @ptr

1. Fetch instruction from [08]
2. Decode instruction
3. Fetch data from mem pointed to by P
4. Add data to [A] and store back in A
5. Set Z flag according to data
6. Set PC to [09]



A=13

Z=0

PC=08

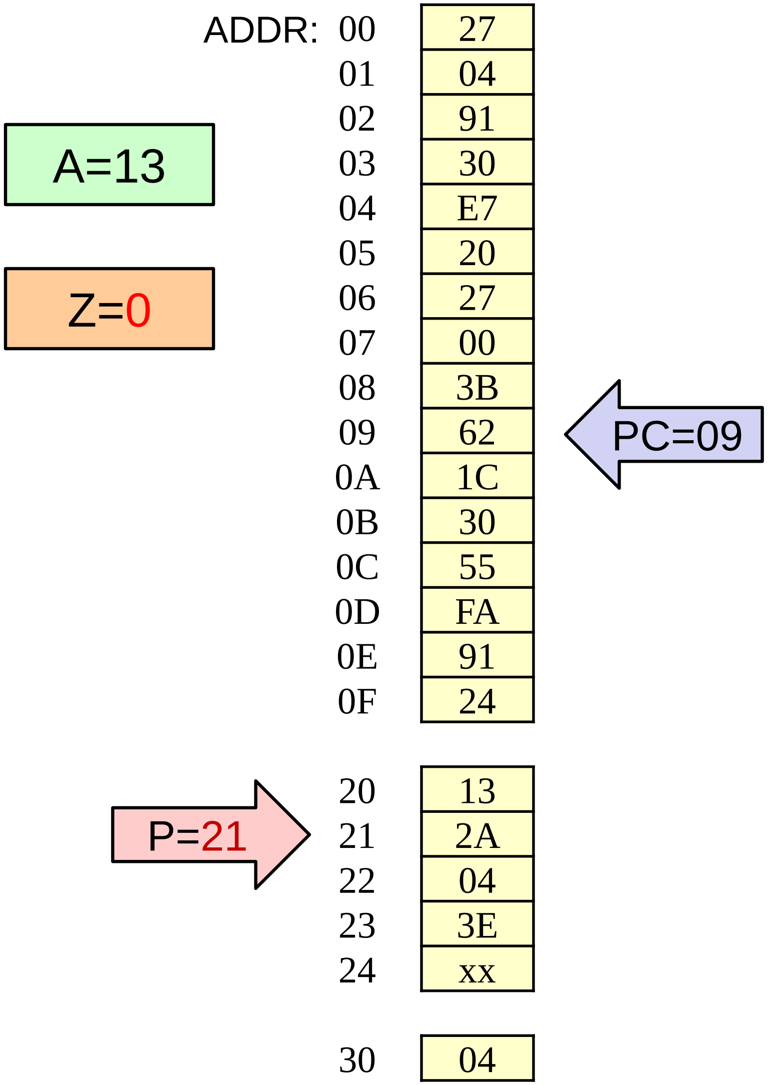
P=20

ADDR:	
00	27
01	04
02	91
03	30
04	E7
05	20
06	27
07	00
08	3B
09	62
0A	1C
0B	30
0C	55
0D	FA
0E	91
0F	24
20	13
21	2A
22	04
23	3E
24	xx
30	04

Instruction 6 (Example)

incp

1. Fetch instruction from [09]
2. Decode instruction
3. Increment pointer P
4. Set Z flag according to data
5. Set PC to [0A]



A=13

Z=0

PC=09

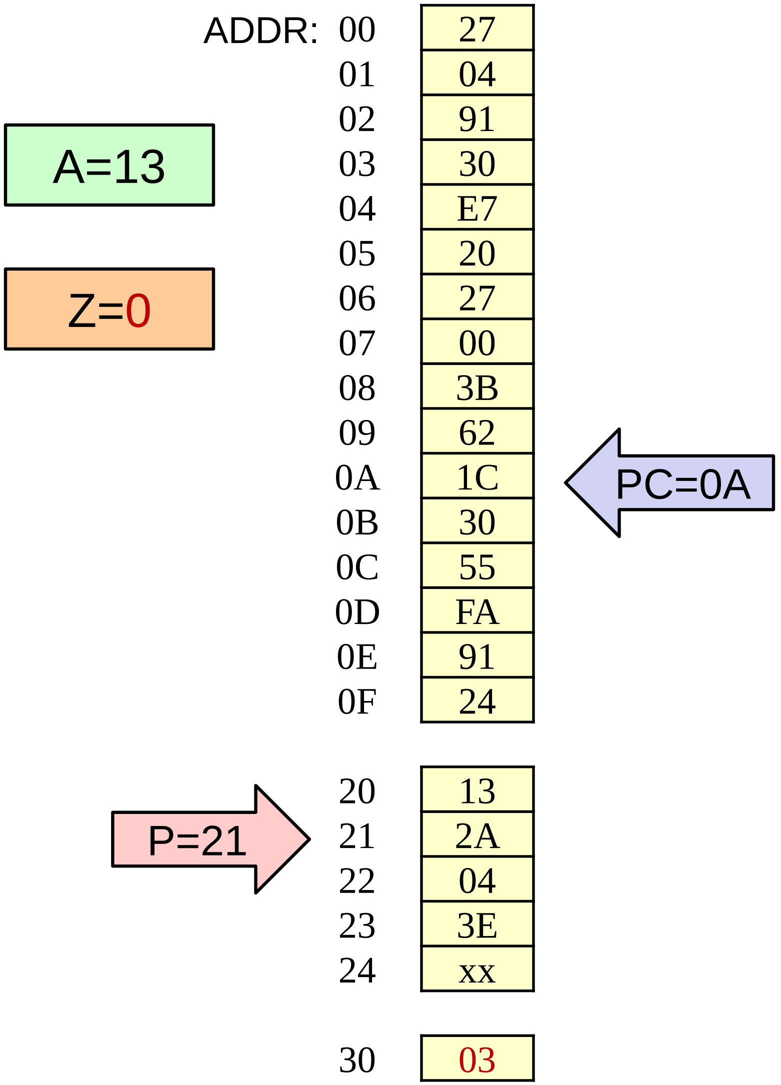
P=21

ADDR:	
00	27
01	04
02	91
03	30
04	E7
05	20
06	27
07	00
08	3B
09	62
0A	1C
0B	30
0C	55
0D	FA
0E	91
0F	24
20	13
21	2A
22	04
23	3E
24	xx
30	04

Instruction 8 (Example)

decm 0x30

1. Fetch instruction from [0A]
2. Decode instruction
3. Fetch address from [0B]
4. Read data from [addr]
5. Decrement data
6. Store data back in addr.
7. Set Z flag according to result
8. Set PC to [0C]



A=13

Z=0

PC=0A

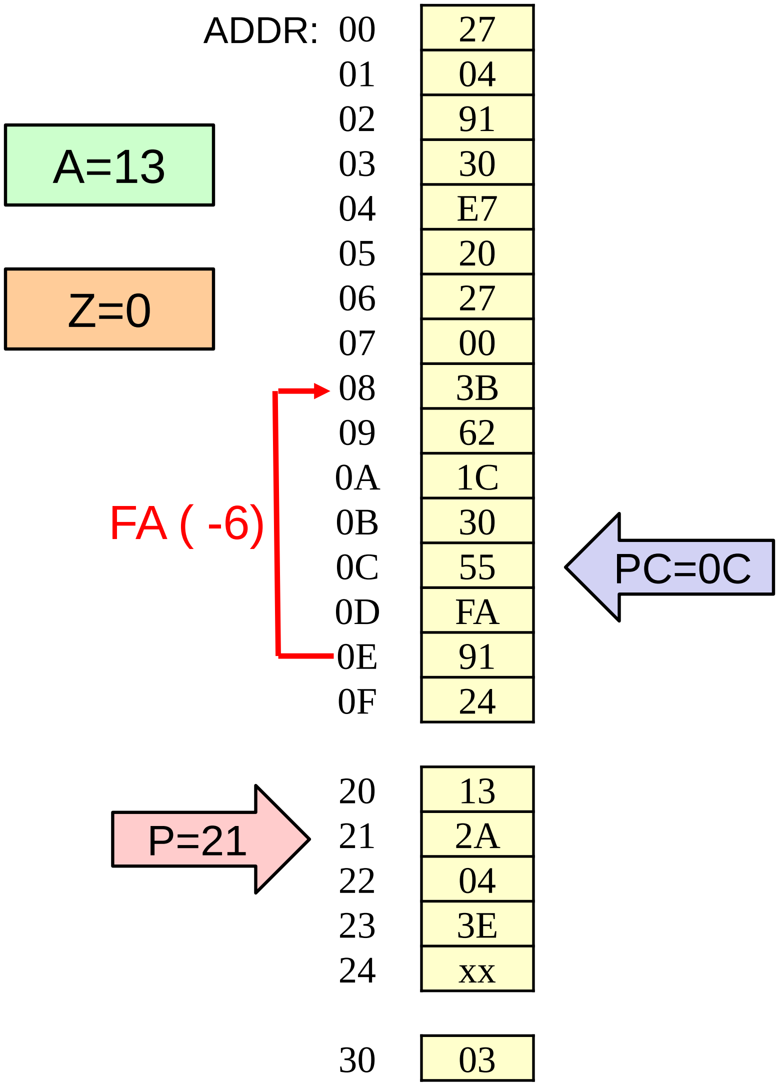
P=21

ADDR:	
00	27
01	04
02	91
03	30
04	E7
05	20
06	27
07	00
08	3B
09	62
0A	1C
0B	30
0C	55
0D	FA
0E	91
0F	24
20	13
21	2A
22	04
23	3E
24	xx
30	03

Instruction 8 (Example)

bnz loop

1. Fetch instruction from [0C]
2. Decode instruction
3. Fetch offset from [0D] (= –6)
4. Set PC to [0E]
5. Test Z flag
6. If Z=0, add offset to PC



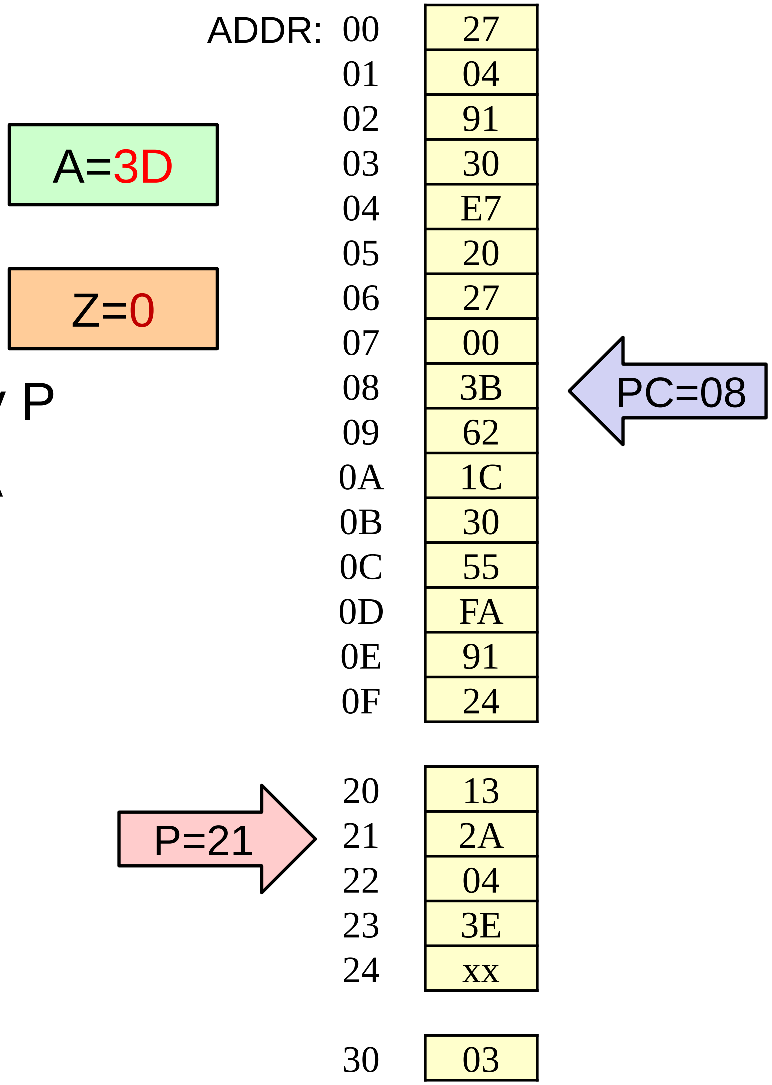
ADDR:	
00	27
01	04
02	91
03	30
04	E7
05	20
06	27
07	00
08	3B
09	62
0A	1C
0B	30
0C	55
0D	FA
0E	91
0F	24
20	13
21	2A
22	04
23	3E
24	xx
30	03

Instruction 9 (Example)

adda @ptr

1. Fetch instruction from [08]
2. Decode instruction
3. Fetch data from mem pointed to by P
4. Add data to [A] and store back in A
5. Set Z flag according to data
6. Set PC to [09]

etc…etc…



A=3D

Z=0

PC=08

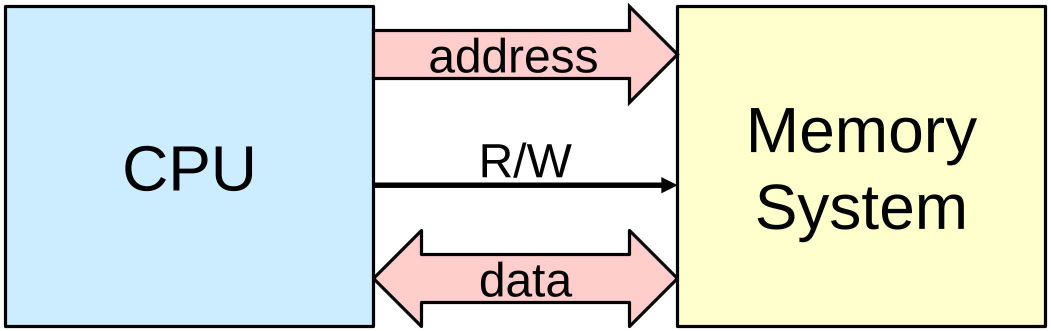
P=21

ADDR:	
00	27
01	04
02	91
03	30
04	E7
05	20
06	27
07	00
08	3B
09	62
0A	1C
0B	30
0C	55
0D	FA
0E	91
0F	24
20	13
21	2A
22	04
23	3E
24	xx
30	03

RISC vs. CISC

- CISC (Complex Instruction Set Computers) support complex instructions that may take many clock cycles to complete:
  - complex instruction format (specific bit positions mean different things in different instructions)
  - rich set of addressing modes
  - many special purpose registers
  - examples include x86, M68000, Pentium, VAX, HCS12
- RISC (Reduced Instruction Set Computers) used simplified instruction set that reduces cycles/instruction and allows increased clock rate.
  - uniform instruction format (faster decoding)
  - load/store architecture with simple addressing modes
  - many general purpose registers
  - more instructions per task
  - simpler to pipeline; simpler to write compiler
  - examples include ARM, SPARC, MIPS

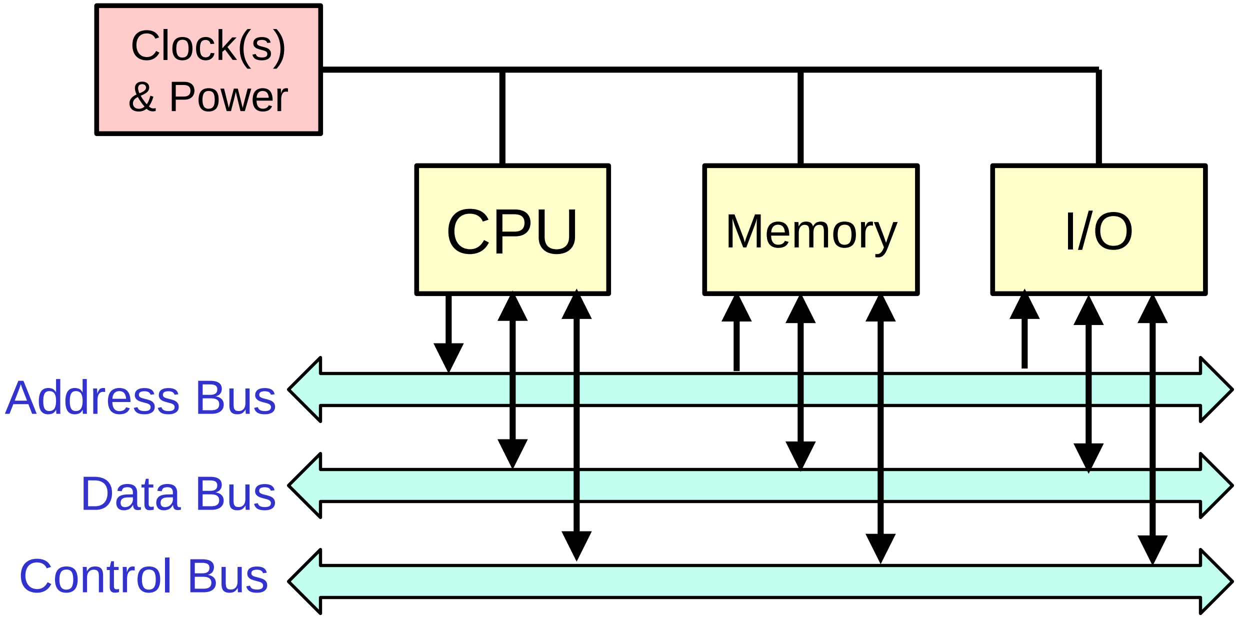
Memory



- Memory is typically a 1-D array of directly addressable locations
- Each location has two components: address and contents
- Each memory loc. can store data, instruction, address, status etc.
- Read (or load) operation transfers data from memory to CPU
- Write (or store) operation transfers data from CPU to memory
- # bits in address field determine memory address space
  - 16 bits allows for 65,636 (64K) distinct memory locations
  - 32 bits allows for 4,294,967,296 (4G) distinct memory locations
- Actual (physical) memory space is usually smaller than memory address space

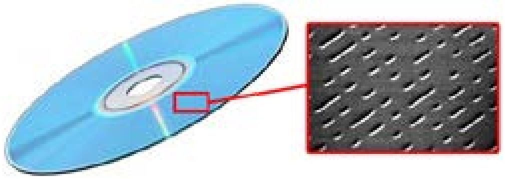
Memory Bus

- Microprocessors typically use one or more buses to communicate with memory and peripherals (I/O)
- Bus is a set of lines (e.g. 16-bit bus) that can be shared by several devices
  - Only one device “writes” (or outputs) data on to bus at any one time – one of more devices can “read” (or input) data that has been written.
- Three buses on HCS12 (control, address and data)



Memory Technologies

- Magnetic Memory
  - earliest form of computer memory
  - magnetic drums, tapes, disks, core memory
  - only magnetic disks in widespread use today
  - non-volatile (retains information when power removed)
  - challenged by flash
- Optical memory
  - Compact Disk (CD) - 700 MB of data
  - Digital Video Disk (DVD) - 4.7 GB of data
  - Blu-Ray Disk (BD) – 25 GB of data
  - non-volatile – longer term storage and archive



- Magnetic and Optical memory are ***not*** random-access
  - highest data rate when data is read sequentially
  - random access requires relatively long seek time
  - not suitable for main memory of computer system

Semiconductor Memory

- Most semiconductor memories are random-access
  - uniform (fast) time to read or write to arbitrary address
  - well suited to main memory of computer system
- Volatile semiconductor memory
  - loses information when power is removed
  - Dynamic Random Access Memory (DRAM)
    - data stored as charge on capacitor
    - each one-bit cell consists of 1 transistor and 1 capacitor
    - highest density
    - read operation is destructive – data must be re-written
    - charge leaks away over time – must be periodically refreshed
  - Static Random Access Memory (SRAM)
    - data stored in cross-coupled inverter
    - 4-6 transistors per cell
    - fastest access times
    - no refresh required

512MB 1Rx16 PC2-5300S-555-12-C3
Made in Korea M470T6464DZ3-CE6 0648
SAMSUNG

Read-Only Memory

- *Why would you want a memory that can only be read?*
- Because ROM is usually non-volatile
  - ideal for holding programs and data that must be available each time power is applied (e.g., boot code, embedded system firmware)
  - some ROM technologies can be erased & rewritten (takes long time)
- Mask Programmed ROM (MROM)
  - programmed at time of manufacture
  - only cost effective in very large quantities
- Programmable ROM (PROM)
  - programmed by user once by blowing fuses (or anti-fuses)
  - requires special programming equipment

- Erasable Programmable ROM (EPROM)
  - Erased by UV light and reprogrammed electrically many times
- Electrically Erasable Programmable PROM (EEPROM)
  - electrically erased and reprogrammed many times
  - can be erased one location, one row or whole chip in one operation

Flash Memory

- Relatively new technology
  - NOR 1984, NAND 1987
- Non volatile – uses trapped charge in floating gate transistor
- Density and speed comparable to hard disk
- Random access read
- Block erase followed by random access write
- Intelligent controllers provide an interface that mimics RAM of hard disk controller
- Ideal for storing slowly changing user data that must survive power outage

Computer Software

- Computer programs are frequently called software
- Software can be written at different levels of abstraction:
- **Machine Code Instructions**
  - lowest level: sequence of bits that are stored in memory
  - directly executed by CPU according to instruction set architecture

address *n*:	1 0 0 1 1 0 1 1
address *n+1*:	0 0 0 0 0 0 1 1

which when executed adds contents of memory address 3 to accumulator A

- **Assembly Language (Assembly Code)**
  - mnemonic representation of machine instructions
  - usually one-to-one correspondence
  - must be translated by an assembler into machine code for execution
  - for example: ADDA $03

High Level Language

- Programming in assembler (assembly language) requires programmer to understand in detail the instruction set architecture of the microprocessor
- Provides maximum control of resources
  - fastest, most compact code
- Very time consuming with long write/execute/debug cycle
- High level languages (e.g. C) allow programmer to work at more abstract level (variables, data structure & operators)
- Programmer does not need to understand details of instruction set architecture
- Compiler (which does understand ISA) converts high level source code to assembly or machine code
- In high level programming environment, output of compiler is often referred to as object code