Computer Architecture & Real-Time Operating System

6. Processor Architecture (1/2)

(Instruction Set Architecture)

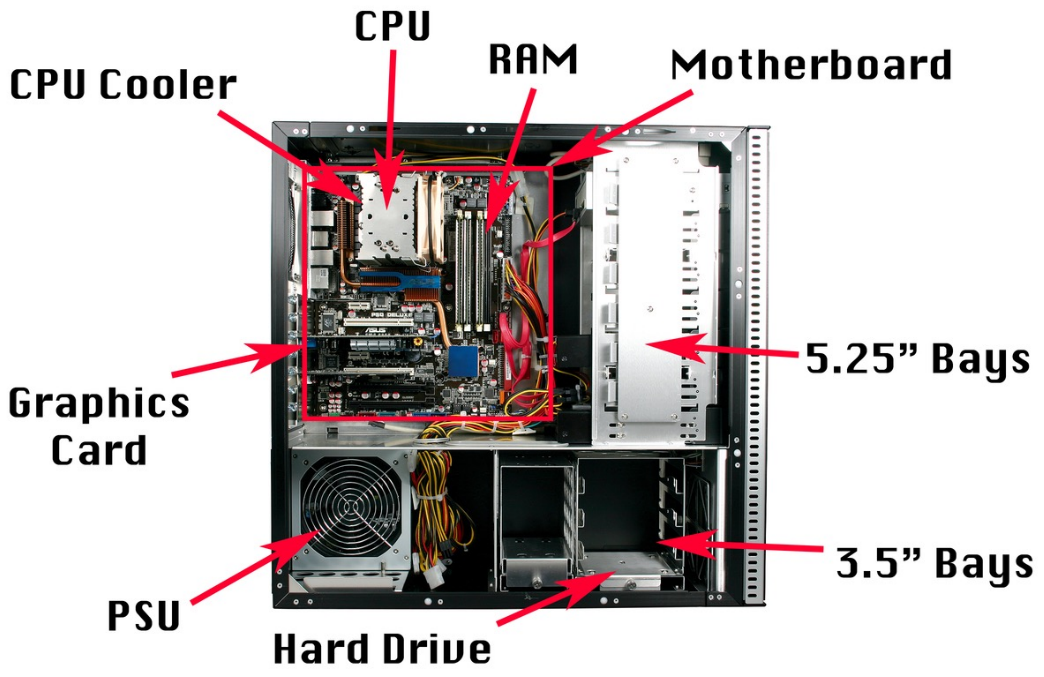
Prof. Jong-Chan Kim

Dept. Automobile and IT Convergence

KMU 국민대학교 KOOKMIN UNIVERSITY

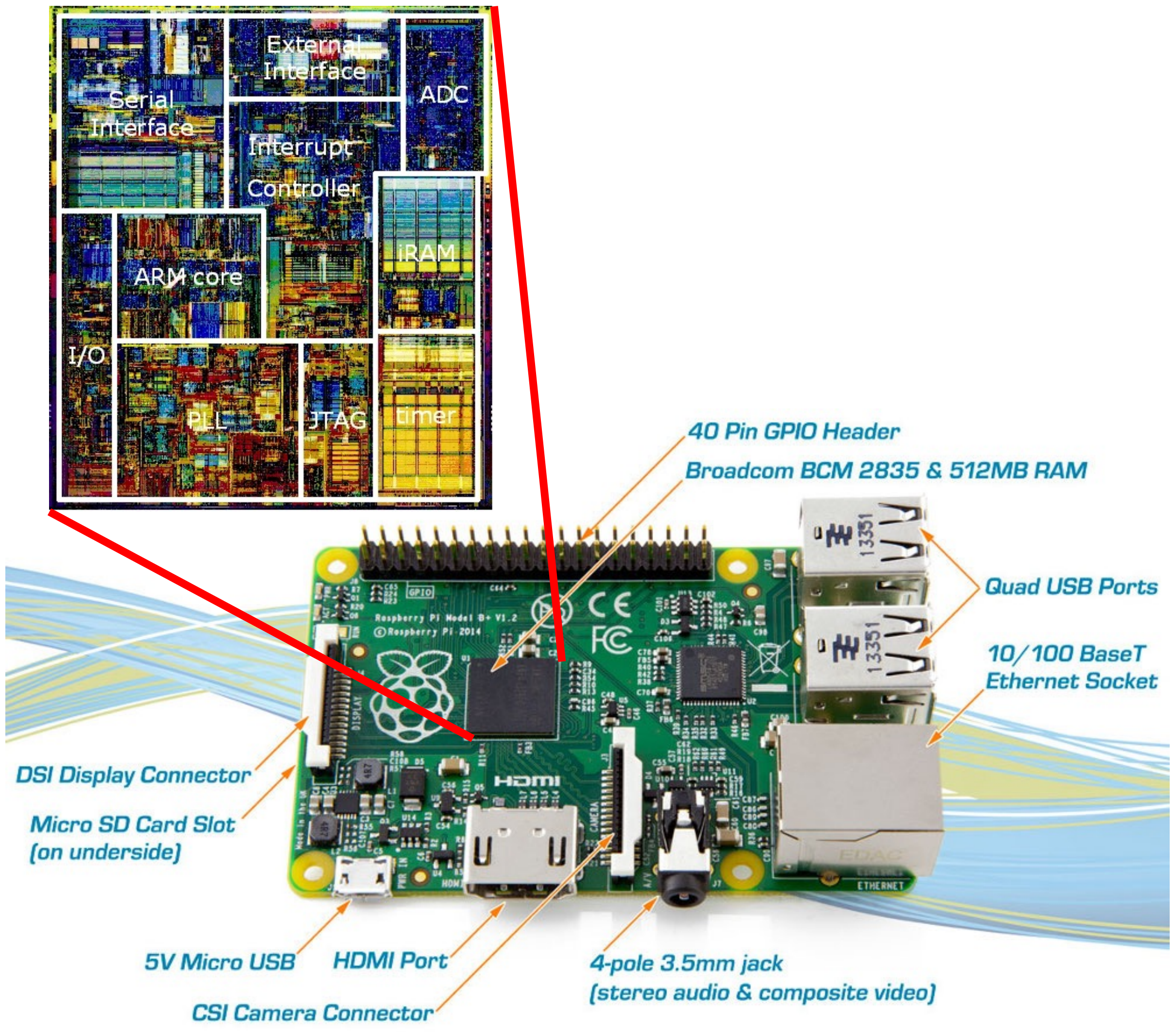
Types of Computer HW Architecture

- Microprocessor or CPU
  - Only computation
- Microcontroller or SoC
  - CPU + Memory + I/O + …



PC Architecture

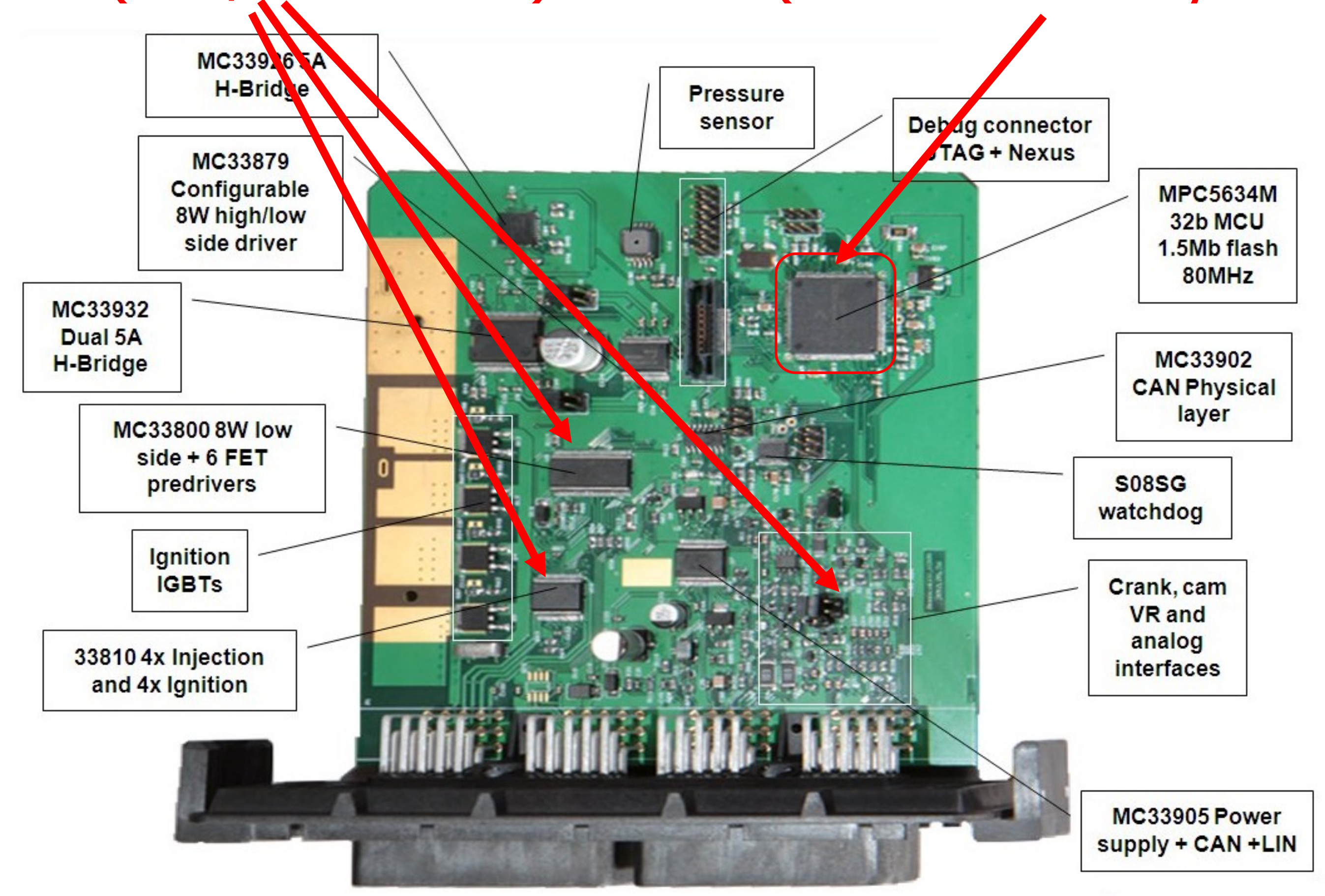
SoC (System on Chip)



SBC (Single Board Computer)

ECU HW Architecture

On-board Peripherals (or I/O Devices) MCU (CPU + Memory + On-Chip Peripherals)


MC33926 5A H-Bridge
MC33879 Configurable 8W high/low side driver
MC33932 Dual 5A H-Bridge
MC33800 8W low side + 6 FET predrivers
Ignition IGBTs
33810 4x Injection and 4x Ignition
Pressure sensor
Debug connector JTAG + Nexus
MPC5634M 32b MCU 1.5Mb flash 80MHz
MC33902 CAN Physical layer
S08SG watchdog
Crank, cam VR and analog interfaces
MC33905 Power supply + CAN +LIN

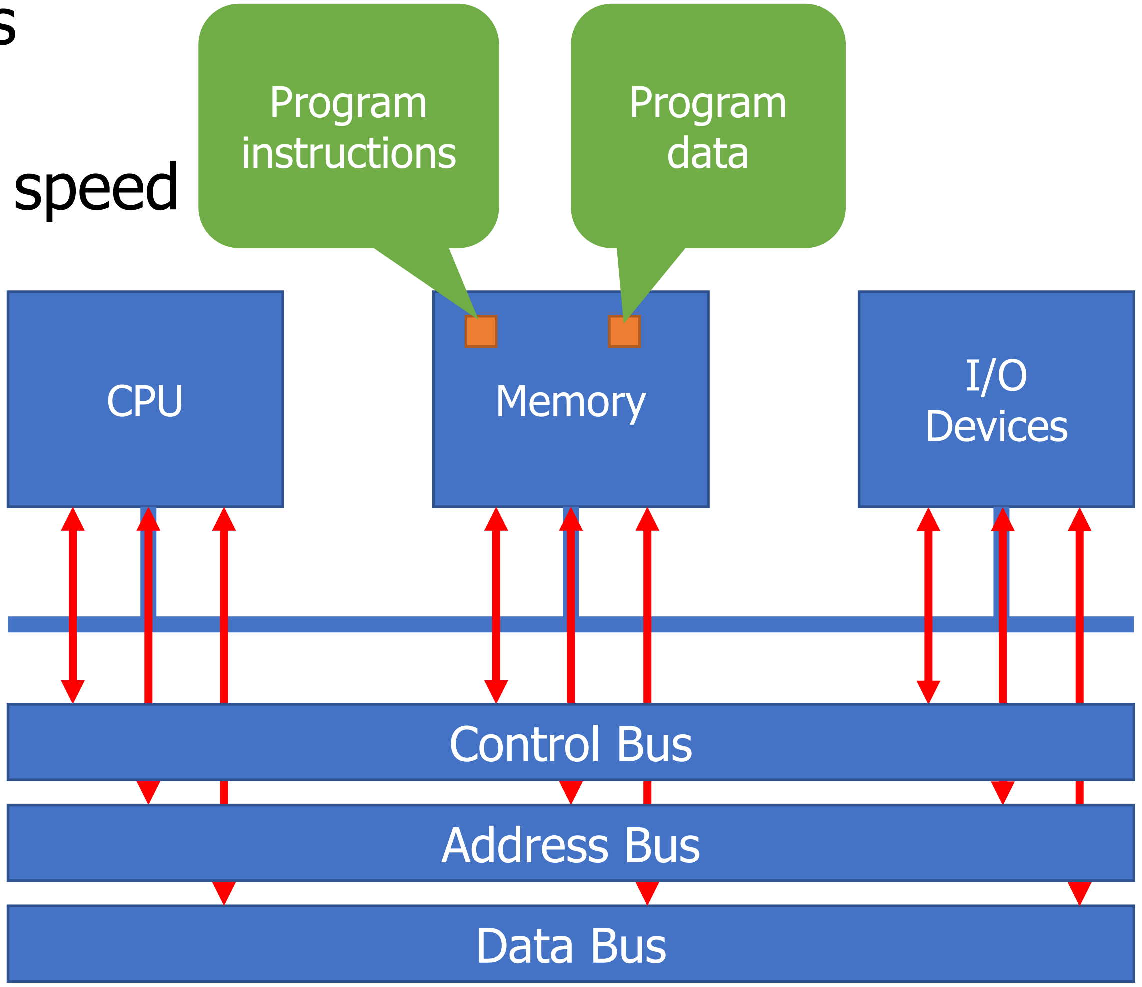
Wire Harness Connector

- GPIO
- ADC
- DAC
- RTC
- Timer
- UART
- …

Bus-based Computer Architecture

- System Bus
  - Connects CPU, Memory, and I/O Devices
  - Bus is a shared medium
  - Bus Bandwidth = Bus width X Bus clock speed

- System bus is similar to scoreboard
  - Everybody can see it at the same time
  - Synchronized by innings like CPU clocks



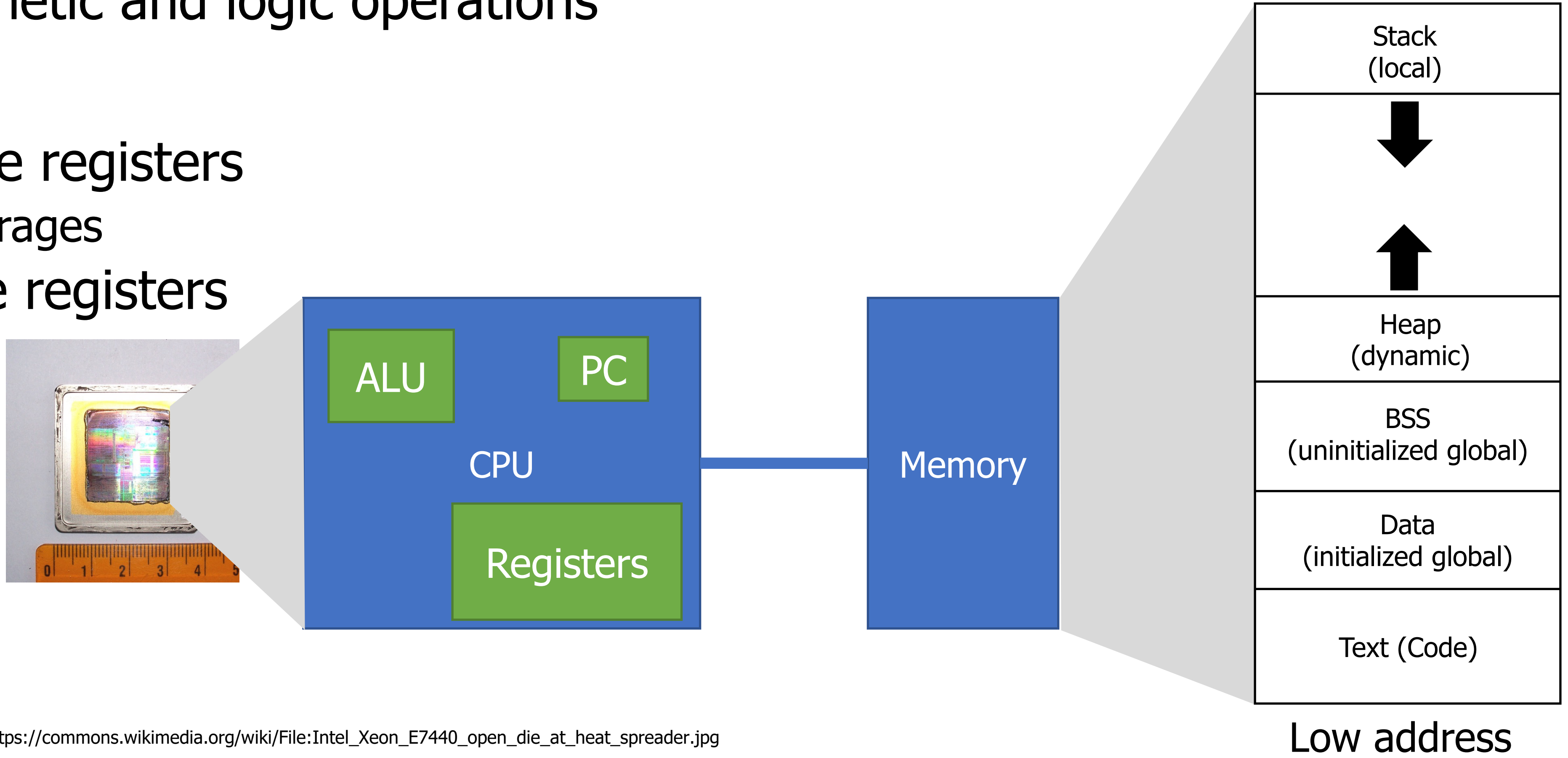
https://varsityscoreboards.com/scoreboards/baseball-scoreboards/3359.html

https://en.wikipedia.org/wiki/System_bus

Inside a CPU

- PC (Program Counter)
  - Indicates the address of the next instruction
- ALU (Arithmetic Logic Unit)
  - Conducts arithmetic and logic operations
- Registers
  - General-purpose registers
    - Temporary storages
  - Special-purpose registers
    - Controllers

One of special-purpose registers

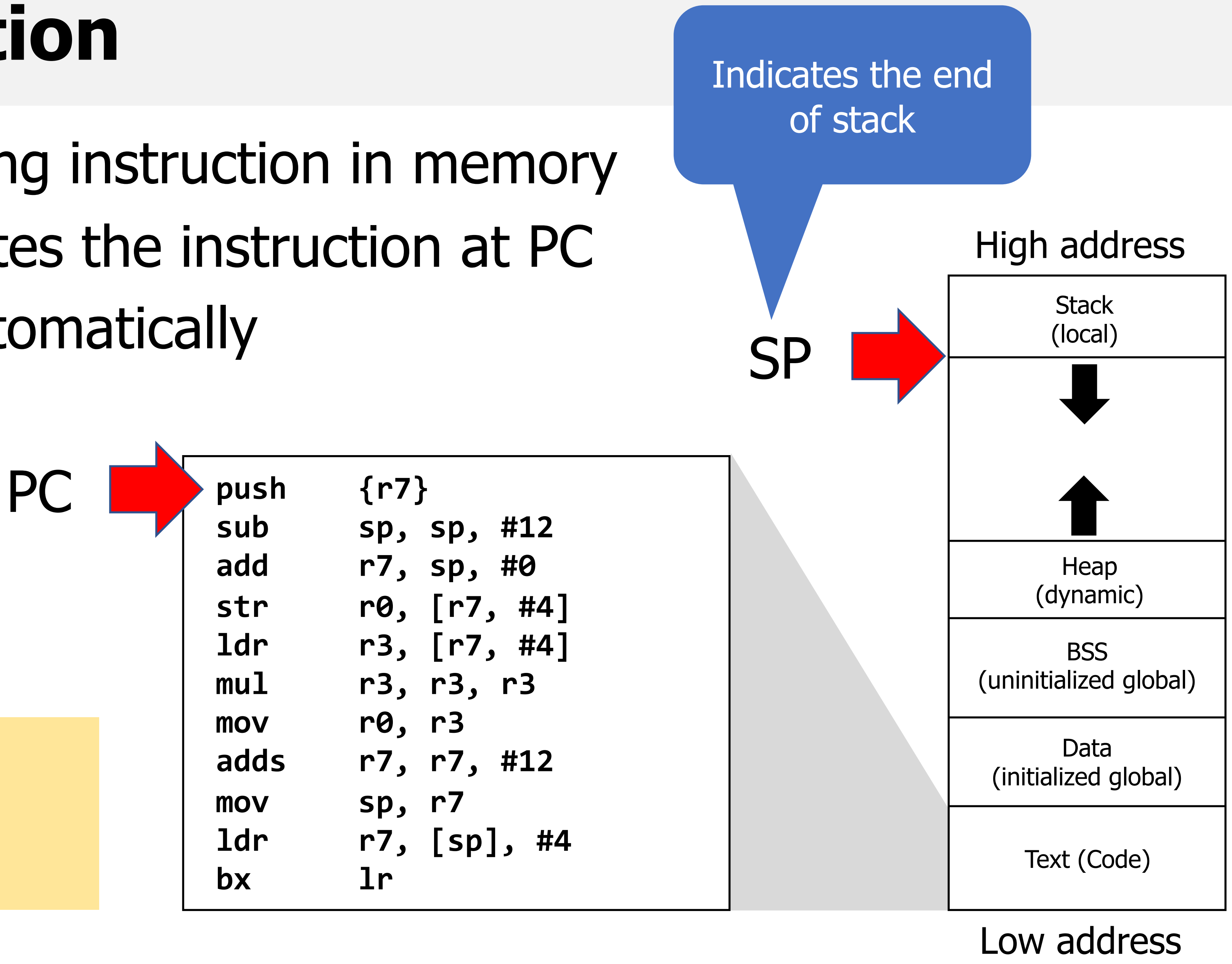


https://commons.wikimedia.org/wiki/File:Intel_Xeon_E7440_open_die_at_heat_spreader.jpg

Program Execution

- Set PC to the beginning instruction in memory
- CPU reads and executes the instruction at PC
- PC is incremented automatically

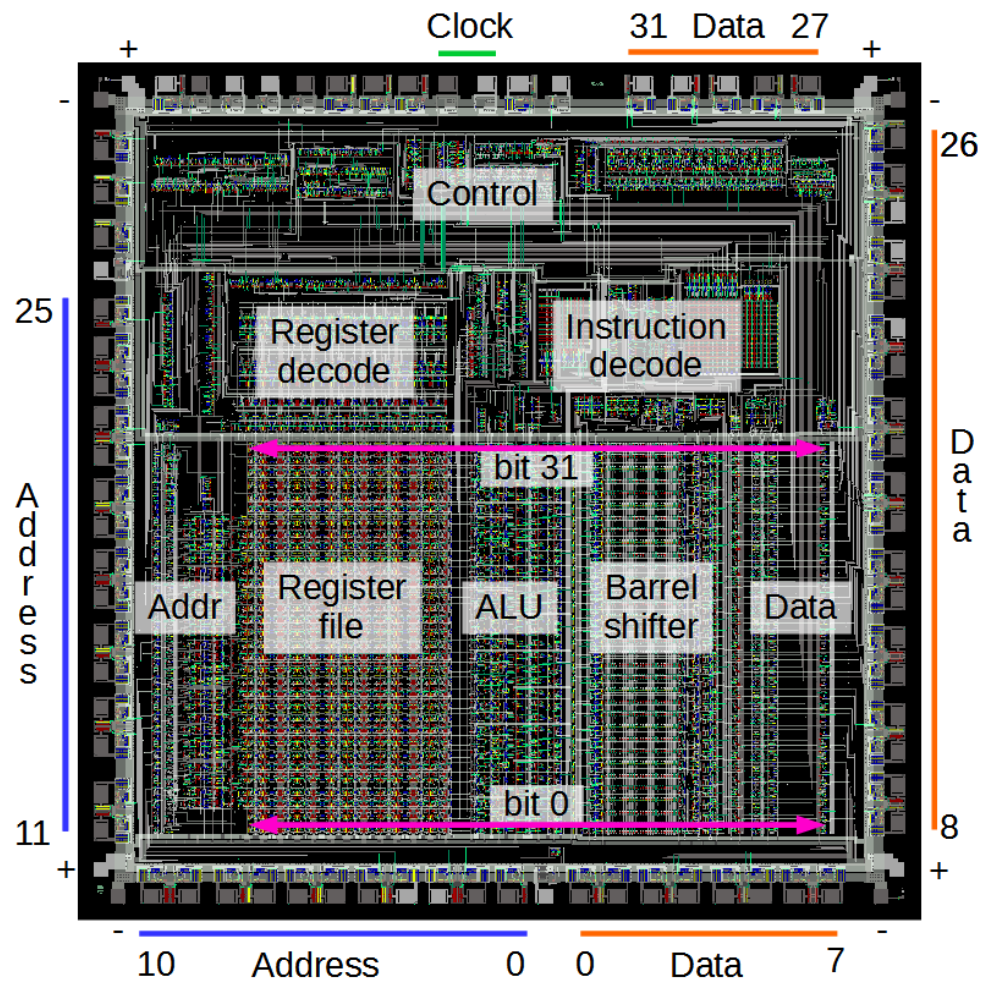
```c
int square(int num) {
    return num * num;
}
```


Indicates the end of stack
SP
PC
High address
Stack (local)
Heap (dynamic)
BSS (uninitialized global)
Data (initialized global)
Text (Code)
Low address

```
push    {r7}
sub     sp, sp, #12
add     r7, sp, #0
str     r0, [r7, #4]
ldr     r3, [r7, #4]
mul     r3, r3, r3
mov     r0, r3
adds    r7, r7, #12
mov     sp, r7
ldr     r7, [sp], #4
bx      lr
```

Registers

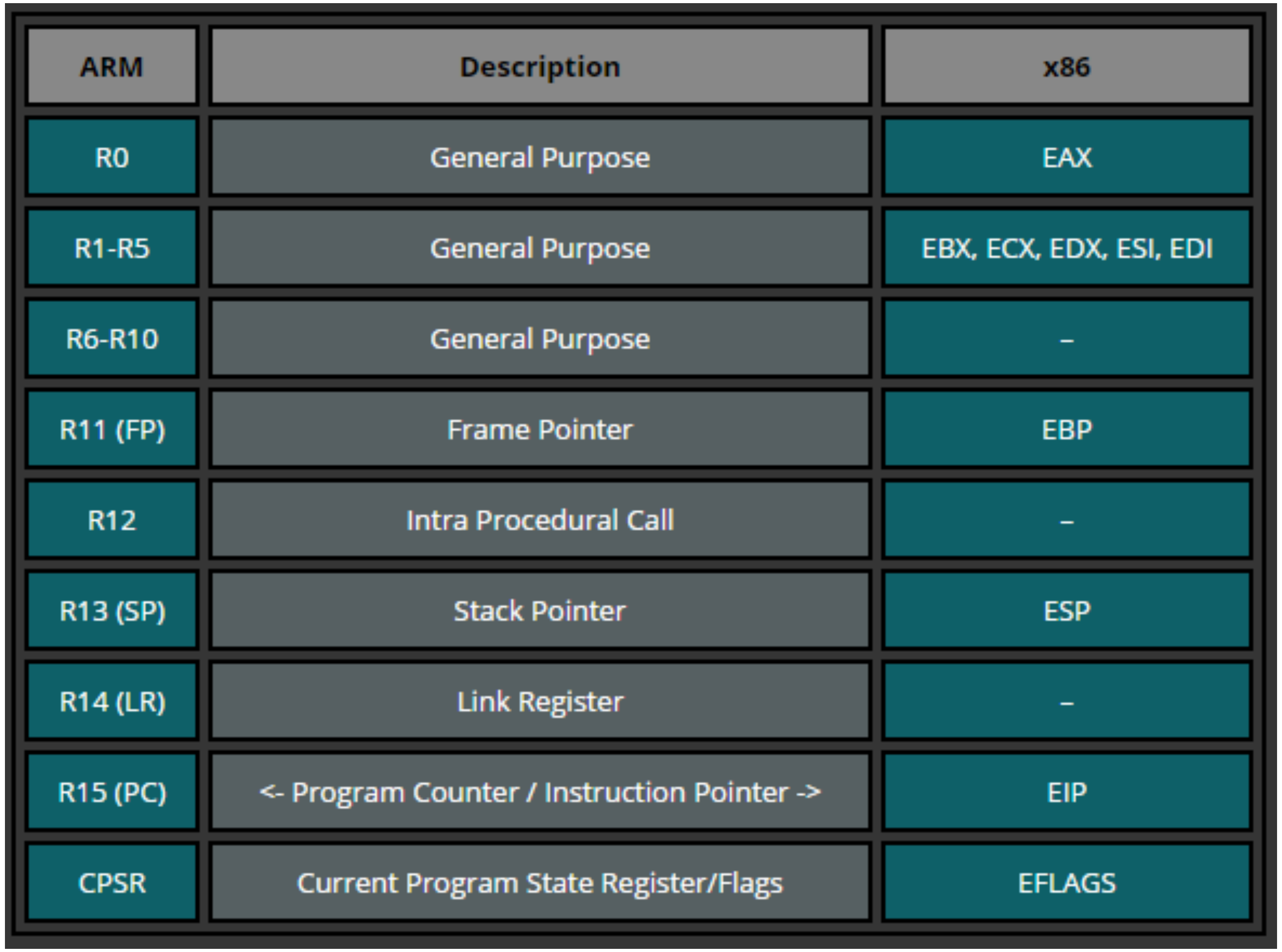
- Temporary storages **inside** CPU
  - Extremely fast compared to memory access
  - Very scarce (only limited number of registers)
- Some CPU registers have special functions
  - PC (Program Counter)
  - SP (Stack Pointer)
  - …



http://www.righto.com/2015/12/reverse-engineering-arm1-ancestor-of.html

X86 and ARM Registers

- Different CPU architectures have different register sets



ARM	Description	x86
R0	General Purpose	EAX
R1-R5	General Purpose	EBX, ECX, EDX, ESI, EDI
R6-R10	General Purpose	–
R11 (FP)	Frame Pointer	EBP
R12	Intra Procedural Call	–
R13 (SP)	Stack Pointer	ESP
R14 (LR)	Link Register	–
R15 (PC)	<- Program Counter / Instruction Pointer ->	EIP
CPSR	Current Program State Register/Flags	EFLAGS

https://azeria-labs.com/arm-data-types-and-registers-part-2/

Processor Architecture

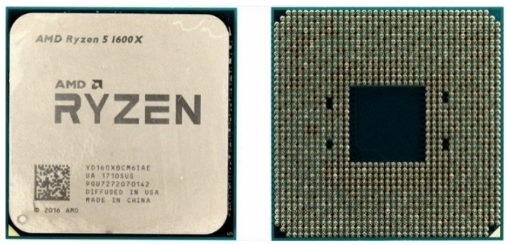
- Instruction Set Architecture (ISA)
  - What CPU understands
- Microarchitecture
  - How CPU is designed

If you completely understand these books, you can be a human CPU

AMD x86-64 Architecture PROGRAMMER'S MANUAL
Volume 1 Application Programming
Volume 2 System Programming
Volume 3 General-Purpose and System Instructions
Volume 4 128-Bit Media Instructions
Volume 5 64-Bit Media and x87 Floating-Point Instructions

Books describing x86-64 ISA (2002)

Intel Core i7



AMD RYZEN

SAMSUNG Exynos

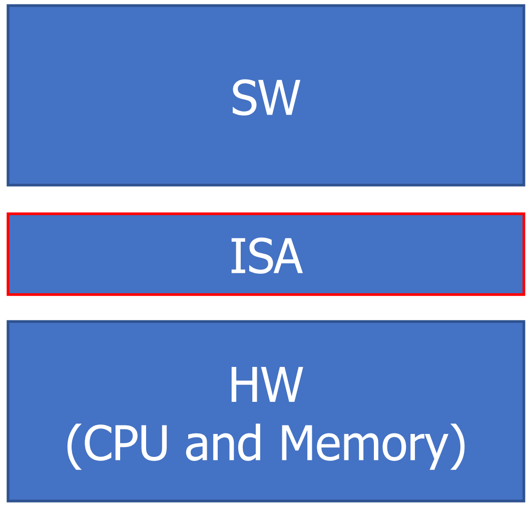
Qualcomm Snapdragon

Understand X86-64 ISA

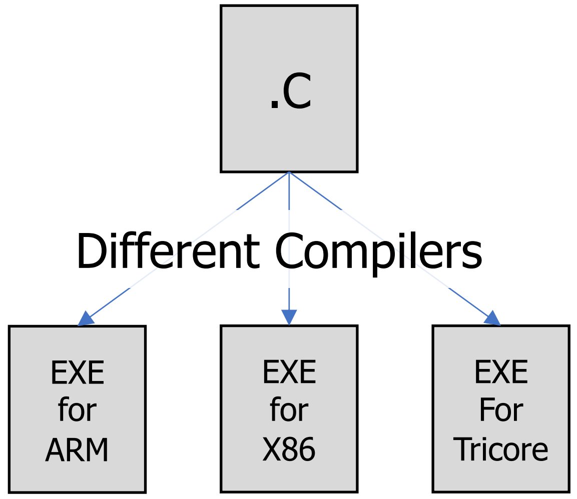
Understand ARM ISA

Instruction Set Architecture (ISA)

- The interface between HW and SW
  - Instructions
  - Registers
  - Memory access mode
  - Endianness (Little-endian, Big-endian, and Bi-endian)
  - …

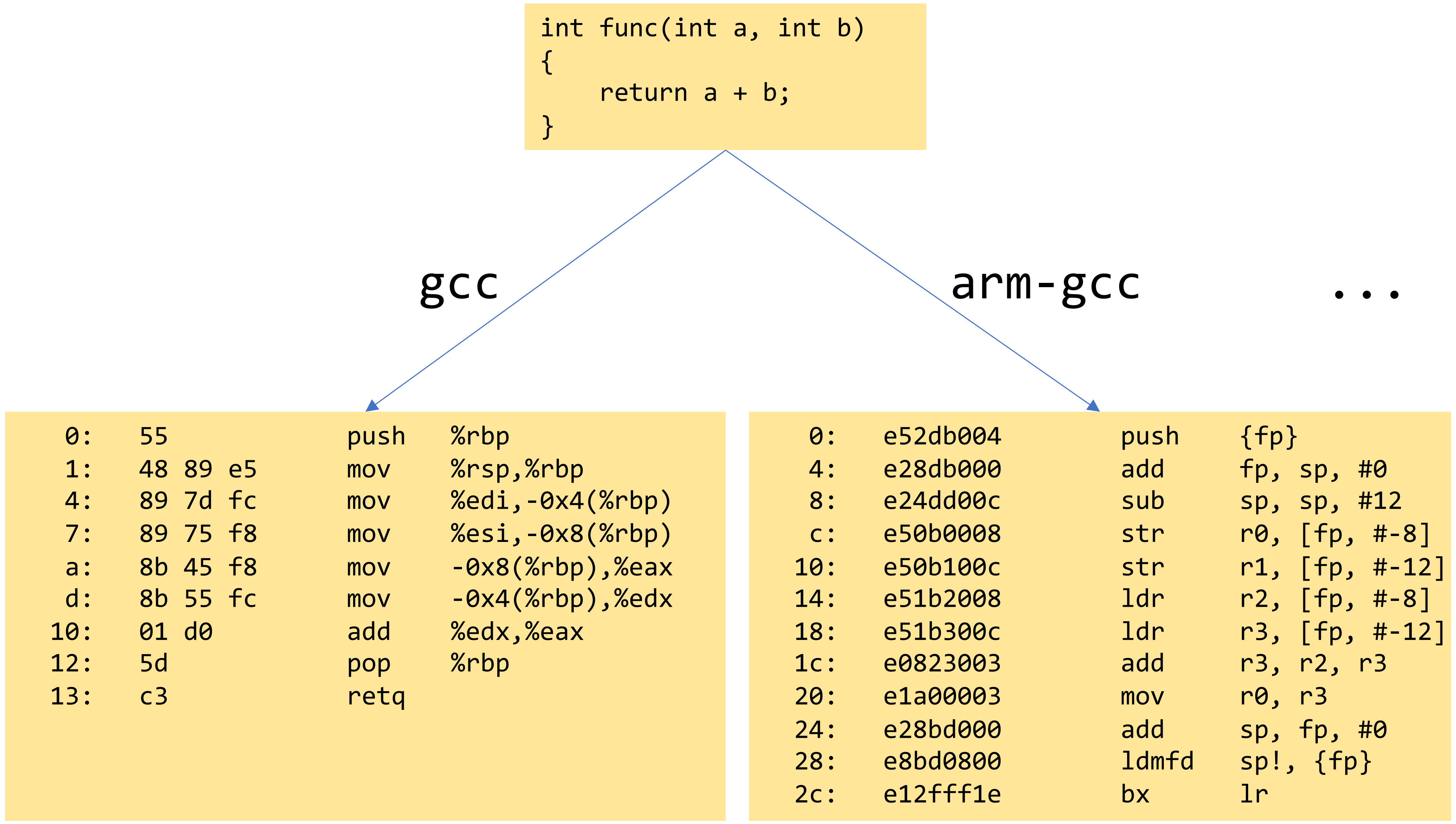


- Different compilers for various ISAs
  - Same C code, but different instructions
  - Compiler developers should understand ISAs completely
  - Host computer's ISA has nothing to do with the target ISA
  - Cross compilation
    - Host ISA ≠ target ISA



Different EXEs for different ISAs

Compilers for Different ISAs



```
int func(int a, int b)
{
    return a + b;
}
```

```
  0:   55                push   %rbp
  1:   48 89 e5          mov    %rsp,%rbp
  4:   89 7d fc          mov    %edi,-0x4(%rbp)
  7:   89 75 f8          mov    %esi,-0x8(%rbp)
  a:   8b 45 f8          mov    -0x8(%rbp),%eax
  d:   8b 55 fc          mov    -0x4(%rbp),%edx
 10:   01 d0             add    %edx,%eax
 12:   5d                pop    %rbp
 13:   c3                retq
```

X86 assembly instructions

```
  0:   e52db004     push    {fp}
  4:   e28db000     add     fp, sp, #0
  8:   e24dd00c     sub     sp, sp, #12
  c:   e50b0008     str     r0, [fp, #-8]
 10:   e50b100c     str     r1, [fp, #-12]
 14:   e51b2008     ldr     r2, [fp, #-8]
 18:   e51b300c     ldr     r3, [fp, #-12]
 1c:   e0823003     add     r3, r2, r3
 20:   e1a00003     mov     r0, r3
 24:   e28bd000     add     sp, fp, #0
 28:   e8bd0800     ldmfd   sp!, {fp}
 2c:   e12fff1e     bx      lr
```

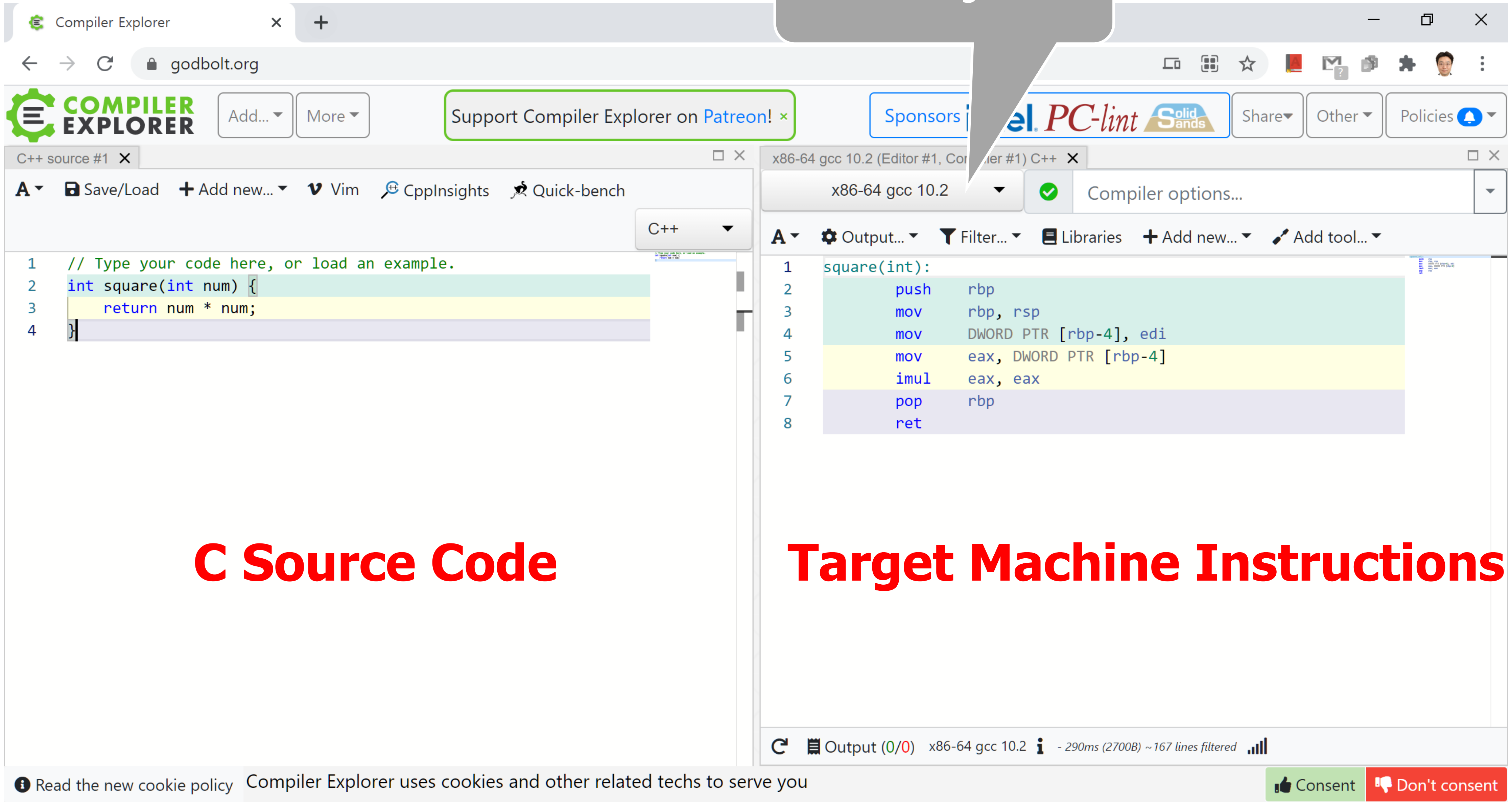
ARM assembly instructions

Installing ARM Compiler

```
# install
$ sudo apt update
$ sudo apt install gcc-arm-none-eabi

# compile
$ arm-none-eabi-gcc -c hello.c
$ arm-none-eabi-objdump –D hello.o
$ arm-none-eabi-gcc --specs=nosys.specs -o prog hello.o
```

Compiler Explorer



```c
// Type your code here, or load an example.
int square(int num) {
    return num * num;
}
```

```
square(int):
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-4], edi
        mov     eax, DWORD PTR [rbp-4]
        imul    eax, eax
        pop     rbp
        ret
```

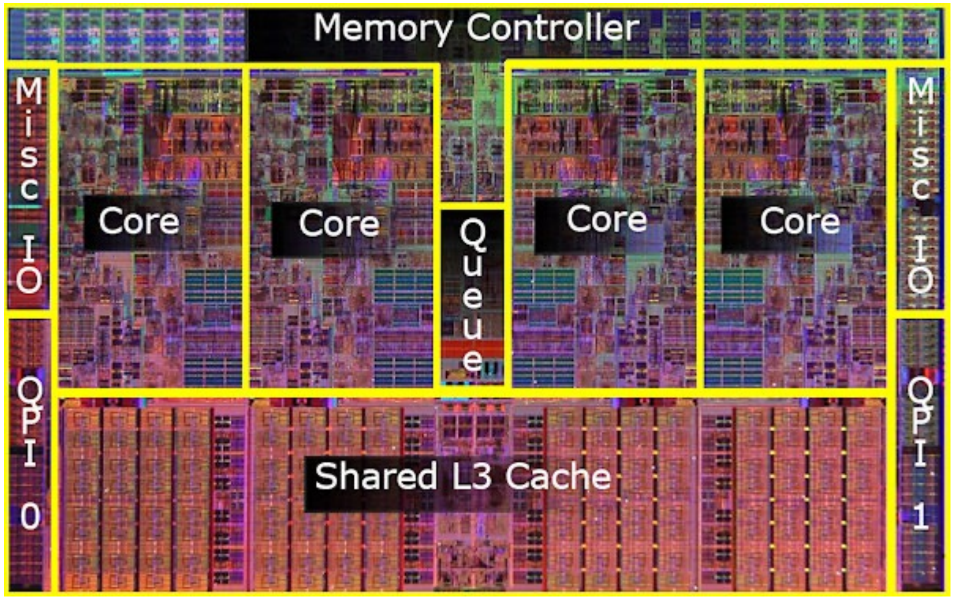
Select a Target ISA

C Source Code

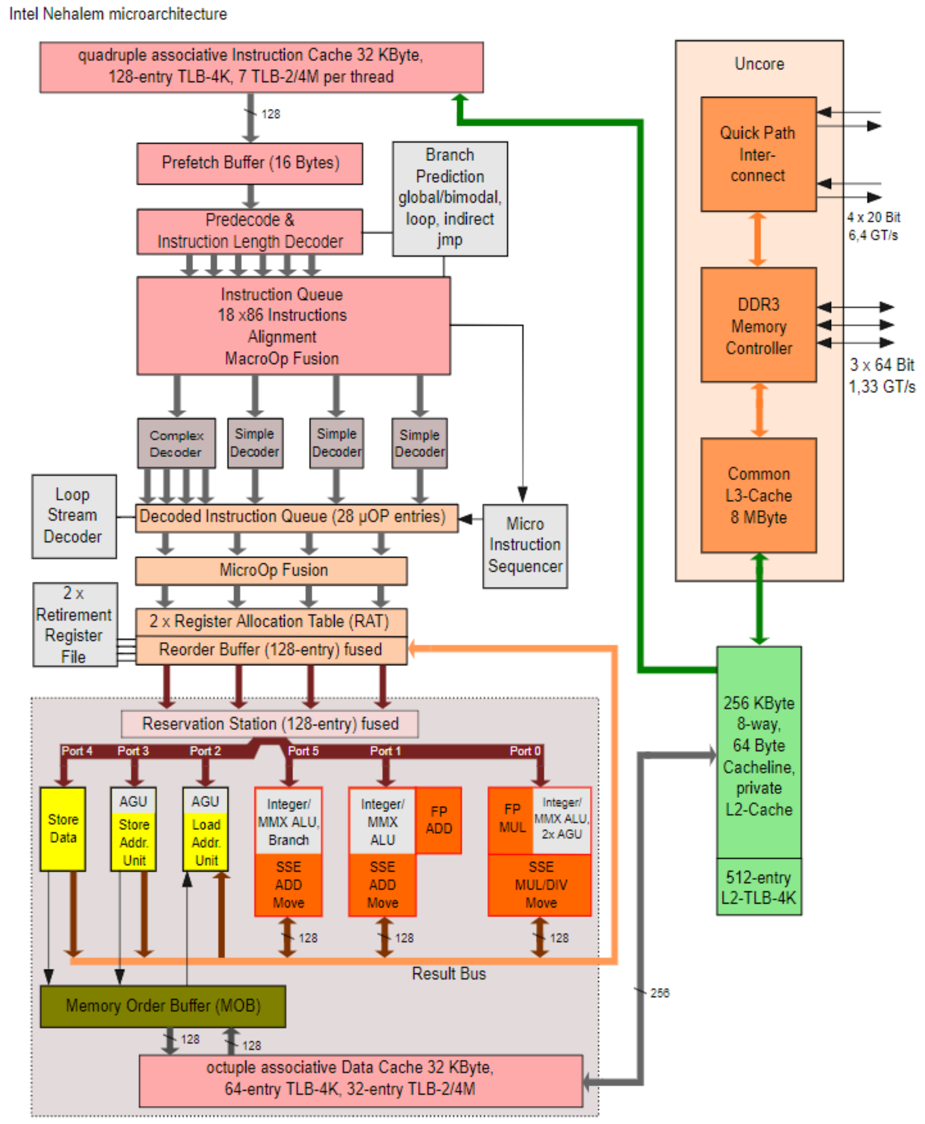
Target Machine Instructions

Microarchitecture

- Chip-level Design
  - Cache
  - Pipelining
  - Out-of-order execution
  - …



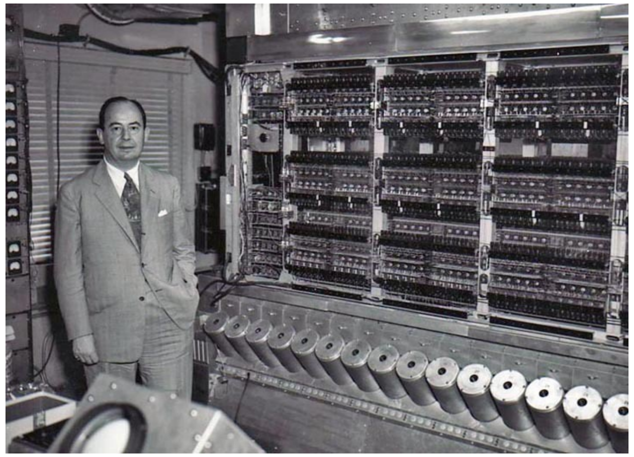
Intel Core i7 (Nehalem) die

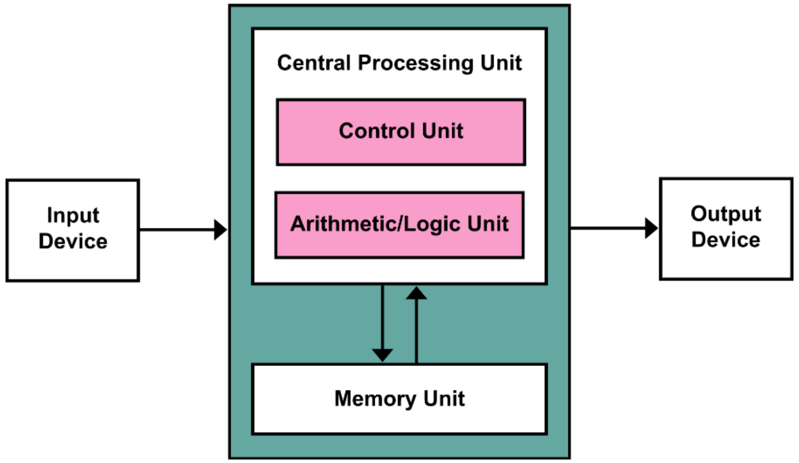

Intel Nehalem microarchitecture
quadruple associative Instruction Cache 32 KByte, 128-entry TLB-4K, 7 TLB-2/4M per thread
128
Prefetch Buffer (16 Bytes)
Branch Prediction global/bimodal, loop, indirect jmp
Predecode & Instruction Length Decoder
Instruction Queue 18 x86 Instructions Alignment MacroOp Fusion
Complex Decoder
Simple Decoder
Simple Decoder
Simple Decoder
Loop Stream Decoder
Decoded Instruction Queue (28 µOP entries)
Micro Instruction Sequencer
MicroOp Fusion
2 x Retirement Register File
2 x Register Allocation Table (RAT)
Reorder Buffer (128-entry) fused
Reservation Station (128-entry) fused
Port 4
Port 3
Port 2
Port 5
Port 1
Port 0
Store Data
AGU Store Addr. Unit
AGU Load Addr. Unit
Integer/ MMX ALU, Branch
SSE ADD Move
Integer/ MMX ALU
FP ADD
SSE ADD Move
FP MUL
Integer/ MMX ALU, 2x AGU
SSE MUL/DIV Move
128
128
128
Result Bus
Memory Order Buffer (MOB)
128
128
octuple associative Data Cache 32 KByte, 64-entry TLB-4K, 32-entry TLB-2/4M
256
Uncore
Quick Path Inter-connect
4 x 20 Bit 6,4 GT/s
DDR3 Memory Controller
3 x 64 Bit 1,33 GT/s
Common L3-Cache 8 MByte
256 KByte 8-way, 64 Byte Cacheline, private L2-Cache
512-entry L2-TLB-4K

Microarchitecture of a processor core

Father of Modern Computer Architecture

- John Von Neumann (1903 ~ 1957)
  - Hungarian-american genius
  - Search "야공만 폰노이만"
- First Draft of a Report on the EDVAC (1945)
  - Stored program concept
  - Instructions and data in the same memory

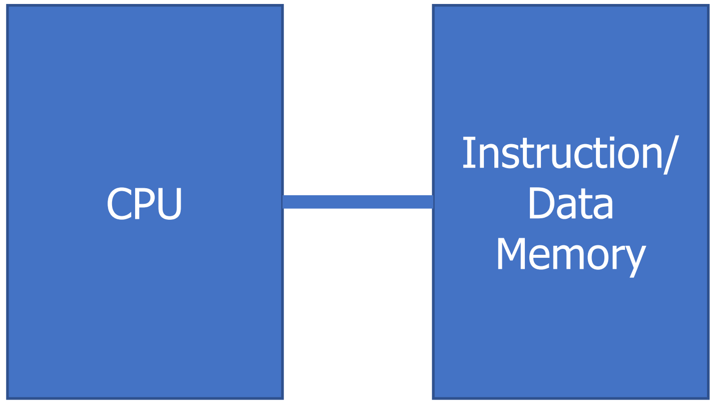

Von Neumann



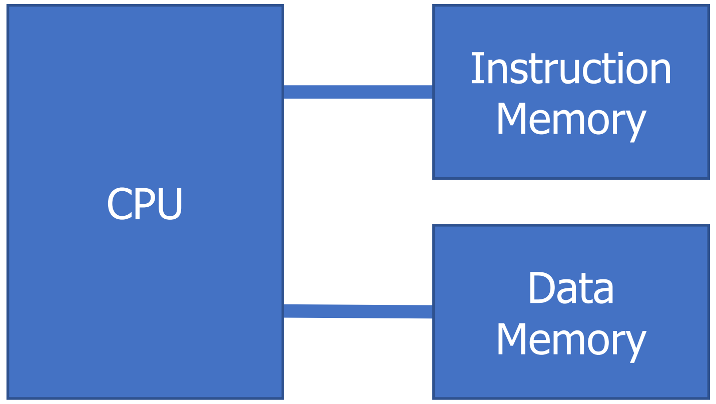
Von Neumann Architecture

Von Neumann vs Havard Architecture

Von Neumann Architecture



Havard Architecture



- Named after John Von Neumann
- One memory for both instructions and data
- No simultaneous accesses to instructions and data
- Bottleneck between CPU and memory

- Named after Havard Mark I computer
- Two separate memories for instructions and data
- Simultaneous accesses to instructions and data
- Less bottleneck between CPU and memory

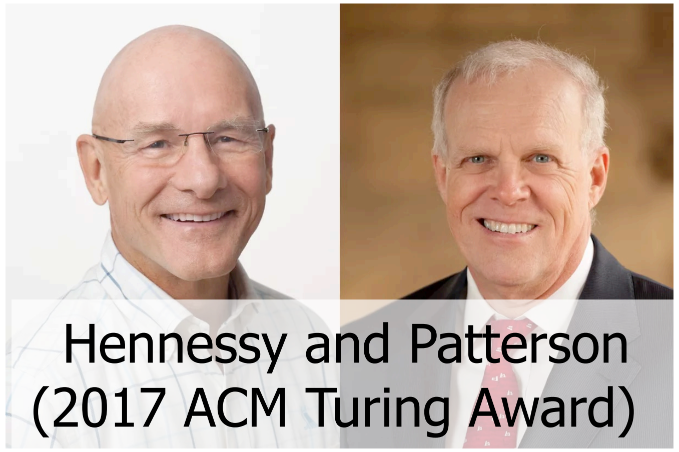
Modern processors have unified memory but separate data path by separate instruction and data caches, which is a combination of Von Neumann and Havard architectures.

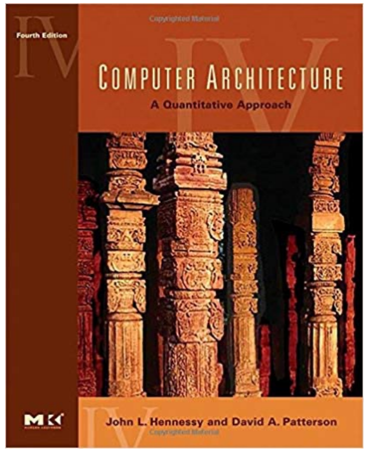
Source: http://www.slideshare.net/pradeepshankhwar/processors-selection-37226806

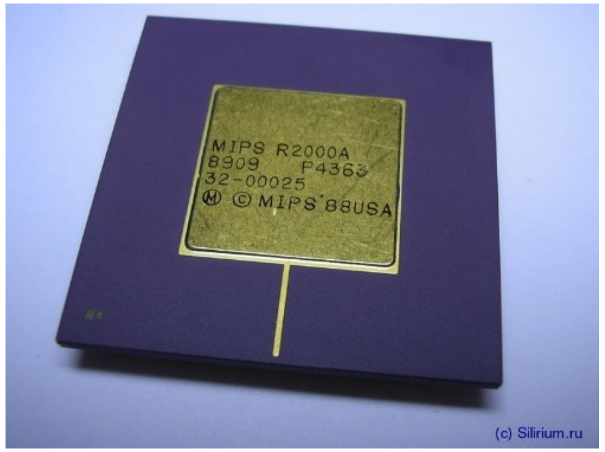
Two Competing Paradigms when Designing ISAs

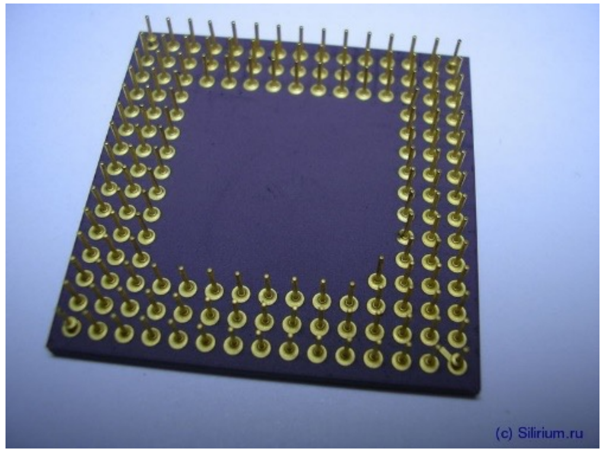
- CISC (Complex Instruction Set Architecture)
  - X86 is a typical example
- RISC (Reduced Instruction Set Architecture)
  - ARM and MIPS are typical examples

- Birth of RISC
  - MIPS R2000 was the first commercial RISC CPU (1986)


Hennessy and Patterson
(2017 ACM Turing Award)







Basic Ideas behind RISC

- CISC has so many instructions people have requested
- People no longer use machine languages
- Instead, compilers generate machine codes
- Then why do we need so many kinds of instructions?
- Let’s provide fewer instructions that are simple yet fast and emulate complex instructions by combining those instructions

CISC and RISC Comparison

CISC	RISC
• Large number of instructions	• Small number of instructions
• Instruction length is variable	• Instruction length is fixed
• More cycles per instruction	• Less cycles per instruction
• Hardware is complex	• Compiler is complex
• Smaller Code Size	• Larger Code Size
• Minimize the number of instructions per program • With increased number of cycles per instruction	• Reduce the number of cycles per instruction • With increased number instructions per program

Two Memory Access Models

Also called register-register architecture

CISC

- Register-memory architecture
  - Operations can be performed on (or) from memory as well as registers

RISC

- Load-store architecture
  - Operations can be performed only on (or) from registers
  - Three steps (Load; Do; Store)
    - Load values from memory to registers
    - Do an operation with registers
    - Store values from registers to memory

```
c = a + b;
```

```
add [r3], [r1], [r2]
```

Register-memory architecture

```
ldr r4, [r1]
ldr r5, [r2]
add r6, r4, r5
str r6, [r3]
```

Load-store architecture

- The addresses of a, b, and c are stored in r1, r2, and r3, respectively
- [] means memory dereferencing (just like pointer dereferencing)

Program Execution Time

Execution Time

CPU Clock Speed

- RISC reduces
- CISC increases

- RISC increases
- CISC reduces

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{seconds}}{\text{cycle}} \text{ X } \frac{\text{cycles}}{\text{instruction}} \text{ X } \frac{\text{instructions}}{\text{program}}$$

Summary

- Instruction Set Architecture
- Von Neumann Architecture vs Havard Architecture
- CISC Architecture vs RISC Architecture