CACHE – PART 2 By Yash

Registers are small storage locations within the Central Processing Unit (CPU) designed to hold data temporarily for operations. They are the fastest accessible memory locations, and their size determines the data width of a CPU.

Types of Registers

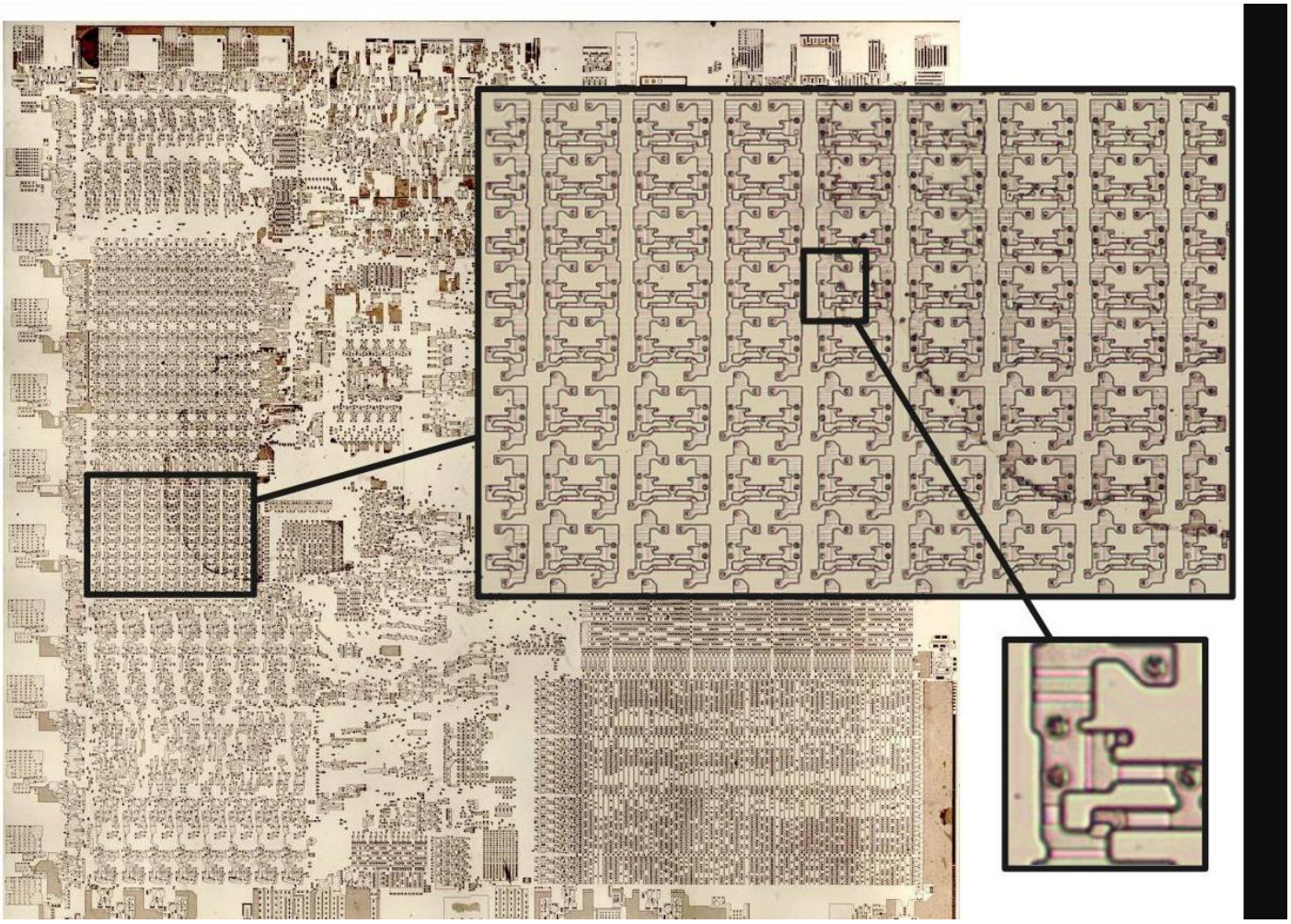
- General-Purpose Registers (GPRs): These are used for both data and addresses. For instance, in the x86 architecture, registers like AX, BX, CX, DX are GPRs.

- Data Registers: These are used to store intermediate data. In ARM processors, R0 to R12 are general-purpose data registers.

- Address Registers: These store addresses and are used to access data and instructions from memory. Examples include Segment Registers in the x86 architecture.

- Special-Purpose Registers:

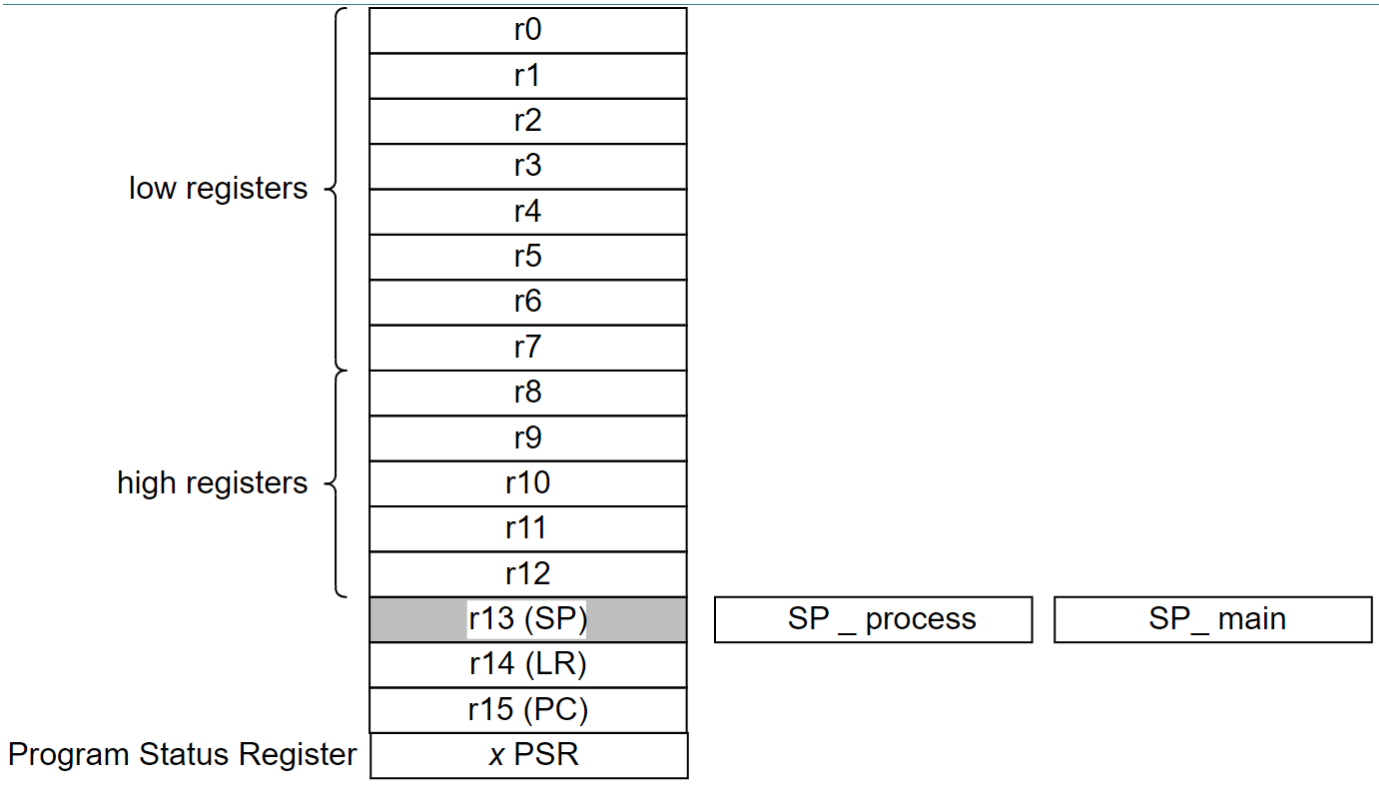
- **Program Counter (PC)**: Holds the address of the next instruction to be executed. If you're running a program with ten instructions, the PC ensures they're executed in order.
- **Instruction Register (IR)**: Contains the current instruction in the CPU, essentially telling the CPU what to do next.
- **Stack Pointer (SP)**: Points to the top of the current stack in memory—a crucial register for function call management in many programming scenarios.
- **Status Register or Flag Register**: Contains flags that provide information about the outcome of operations, like Zero (Z), Carry (C), or Overflow (O) flags.



Die photo of the 8086, zooming in on the lower register file (eight 16-bit registers) and then a single register cell. The metal and polysilicon were removed for this photo to show the silicon structures. (Img src from link below)

A very in detailed explanation of 8086 registers: https://www.righto.com/search?q=registers

A Peak into register of ARM Cortex M4 Microcontroller :



- General-Purpose Registers (GPRs):

The Cortex-M4 contains 13 GPRs (**R0-R12**). These 32-bit registers are versatile and are utilized for a multitude of operations, such as arithmetic, logical operations, and data transfers.

Example:

In ARM Assembly, a simple addition might look like:

MOV R1, #5 ; Load immediate value 5 into R1 MOV R2, #7 ; Load immediate value 7 into R2 ADD R0, R1, R2 ; Add values in R1 and R2, store result in R0

- Data and Address Registers:

The distinction between data and address registers isn't explicitly separated in Cortex-M4. Instead, the general-purpose registers can act both as data and address registers.

Example:

For data load/store operations:

LDR R3, [R2] ; Load data from the address in R2 into R3 STR R3, [R1] ; Store data from R3 into the address in R1

An article about ARM Cortex M4 Core Registers: https://www.linkedin.com/pulse/arm-cortex-m4-core-registers-ahmed-abd-el-ghafar-mohammed

- Special-Purpose Registers:

- Program Counter (PC) - R15:

The PC is auto-incremented after fetching an instruction. However, branching instructions can modify the PC to change the flow of execution.

Example:

For a branch if zero:

BEQ somewhere ; Branch to label 'somewhere' if Zero flag is set

- Link Register (LR) - R14:

The LR holds the return address when a subroutine is called.

Example:

Calling and returning from a subroutine:

BL subroutine ; Branch to 'subroutine' and save return address in LR ... subroutine: ... BX LR ; Return to the address in LR

- Stack Pointer (SP) - R13:

The Cortex-M4 uses the SP for push/pop operations.

Example:

Pushing and popping from the stack:

PUSH {R0-R3} ; Push R0 to R3 onto the stack ... POP {R0-R3} ; Pop values from the stack into R0 to R3

- xPSR:

The xPSR holds condition flags and other status bits.

Example:

After an arithmetic operation, the APSR sub-field of the xPSR might be updated with flags like Zero (Z) or Carry (C).

SUBS R0, R1, R2 ; Subtract R2 from R1, store in R0 and update flags BNE not_zero ; Branch if result is not zero

- Control Register:

This register determines the active stack pointer and mode.

Example:

Switching to use the Process Stack Pointer:

MRS R0, CONTROL ; Read CONTROL register into R0 ORR R0, R0, #2 ; Set the bit to use PSP MSR CONTROL, R0 ; Update CONTROL register

- Floating Point Registers:

These registers handle floating-point operations.

Example:

Floating-point addition:

VADD.F32 S0, S1, S2 ; Add values in S1 and S2, store result in S0

- **Register Width and CPU Architecture**

The width of a register is crucial as it often determines the native word size of a CPU. For instance:

1. 8-bit CPUs: The Dawn of Personal Computing

Intel 8080: Introduced in 1974, the Intel 8080 was one of the first mainstream microprocessors. With its 8-bit registers, it could natively process data in chunks of 8 bits (or 1 byte) at a time.

Registers:

- Accumulator (A): Used for arithmetic and logic operations.
- B, C, D, E, H, L: General-purpose registers.
- Stack Pointer (SP) and Program Counter (PC): Both 16-bit, allowing for larger memory addressing despite the 8-bit data width.

Sample code:

MVI A, 0x32 ; Load immediate value 0x32 into accumulator ADD B ; Add value in B register to accumulator

2. 16-bit CPUs: Doubling Capabilities

Intel 8086: Launched in 1978, this CPU marked Intel's transition into the 16-bit era. Now, data operations could be performed on 16 bits (or 2 bytes) simultaneously.

Registers:

- AX, BX, CX, DX: Primary 16-bit general-purpose registers.
- SI, DI: Source and Destination index registers.
- SP, BP: Stack pointers.
- IP: Instruction Pointer.
- Segment Registers: CS, DS, ES, SS to handle memory segmentation.

Sample Code:

MOV AX, 0x1234 ; Load value 0x1234 into AX ADD AX, BX ; Add value in BX to AX

3. 32-bit CPUs: The Rise of Modern Computing

Intel Pentium Series: The 1990s saw the dominance of 32-bit architectures, with the Pentium series leading the charge. Computers could now handle 32 bits (or 4 bytes) of data in one go.

Registers:

- EAX, EBX, ECX, EDX: Extensions of the original 8086 registers to 32 bits.
- ESI, EDI: Extended source and destination registers.
- ESP, EBP: Extended stack pointers.
- EIP: Extended Instruction Pointer.
- Segment Registers: CS, DS, ES, SS, FS, GS.

Sample Code:

MOV EAX, 0x12345678 ; Load value 0x12345678 into EAX ADD EAX, EBX ; Add value in EBX to EAX

4. 64-bit CPUs: The Current Titans

Intel Core (x86_64): The transition to 64-bit (like the Intel Core series) meant a leap in computational power and memory addressing. Operations on 8 bytes of data became the norm.

Registers:

- RAX, RBX, RCX, RDX: 64-bit extensions of the primary registers.
- RSI, RDI, RSP, RBP, RIP: Extended index, stack, and instruction pointers.
- R8 to R15: New general-purpose registers introduced.
- Segment Registers: CS, DS, ES, SS, FS, GS.

Sample Code:

MOV RAX, 0x1234567890ABCDEF ; Load value into RAX ADD RAX, RBX ; Add value in RBX to RAX

As we've transitioned from 8-bit to 64-bit architectures over the decades, we can process more data at once, allowing for faster computations and larger memory addresses.

- Real-Time Examples

Example 1: When you run a program, the PC (Program Counter) register ensures each instruction runs in sequence. If a program has instructions A, B, and C, the PC ensures A runs before B, and B runs before C.

Example 2: When performing arithmetic operations, the Flag Register comes into play. Suppose you're subtracting two numbers, and the result is zero. The Zero (Z) flag in the Flag Register will be set, indicating this outcome.

Example 3: In embedded systems, like a thermostat, registers play a crucial role. When the thermostat measures temperature, it might store the immediate reading in a register before deciding if the heater should be turned on or off.

- Registers in Modern Architectures

Modern CPUs, especially in multi-core setups, have hundreds of registers. For instance:

- ARM Cortex-A Series: Used in many smartphones, these have 31 GPRs, each 32 bits (for A32 architecture) or 64 bits (for A64 architecture) wide.

1.1 General-Purpose Registers (GPRs):

- **A32 Architecture**: The ARM Cortex-A series, when operating in A32 (or ARMv8-A 32-bit) mode, offers 31 general-purpose registers (**R0** to **R30**), each 32 bits wide. The 32nd register can be either the Stack Pointer (**SP**), the Zero Register (**ZR**), or the Platform Register (**PL**).
- **A64 Architecture**: In A64 mode (or ARMv8-A 64-bit), the series still has 31 general-purpose registers, but each is now 64 bits wide, labeled **X0** to **X30**. Additionally, when these registers are accessed as their 32-bit variants, they are labeled **W0** to **W30**.

Sample Code (ARM Assembly for A64):

MOV X1, #100 ; Load immediate value 100 into X1 ADD X0, X1, X2 ; Add values in X1 and X2, store result in X0

1.2 Special Registers:

- **Program Counter (PC)**: Points to the next instruction.
- **Stack Pointer (SP)**: Used to point to the top of the stack in memory.
- **ELR (Exception Link Register)**: Holds the address to return to after an exception.
- **Floating-Point Registers**: The ARM Cortex-A series has up to 32 64-bit floating-point registers **D0** to **D31** for double-precision. These can be accessed as 128-bit registers **Q0** to **Q31** for SIMD (Single Instruction, Multiple Data) operations.

- x86_64 Architecture: While the original x86 architecture had a limited number of registers, the modern x86_64 architecture (like in Intel Core series) has 16 GPRs, each 64-bits wide, and additional registers for floating-point computations, multimedia operations (MMX, SSE), and more.

2.1 General-Purpose Registers (GPRs):

- The x86_64 architecture expanded the original 8 GPRs to 16, labeled **RAX**, **RBX**, **RCX**, **RDX**, **RBP**, **RSP**, **RSI**, **RDI**, and **R8** to **R15**. Each of these is 64 bits wide. These registers also have 32-bit (**EAX** to **E15**), 16-bit (**AX** to **R15W**), and 8-bit (**AL** to **R15B**) sub-registers.

Sample Code (x86_64 Assembly):

MOV RAX, 50h ; Load hex value 50 into RAX ADD RBX, RAX ; Add values in RAX to RBX

2.2 Special Registers:

- **RIP (Instruction Pointer)**: The 64-bit equivalent of the EIP, pointing to the next instruction.
- **Floating Point Unit (FPU) Registers**: The x86_64 architecture includes 8 80-bit registers for floating-point computations, **ST(0)** to **ST(7)**.
- **MMX Registers**: For multimedia operations, 8 64-bit **MM0** to **MM7** registers exist.
- **SSE Registers**: For Streaming SIMD Extensions, there are 16 128-bit registers **XMM0** to **XMM15**.
- **AVX Registers**: Advanced Vector Extensions expand on SSE, introducing 256-bit **YMM** registers and, in AVX-512, 512-bit **ZMM** registers.

Registers, while minuscule in capacity, play a pivotal role in ensuring the smooth and efficient operation of a CPU. They're the workbenches of the processor, holding data and instructions, guiding operations, and reflecting outcomes. As processors become more advanced, the role and complexity of registers continue to evolve, solidifying their place as the heartbeat of any computing device.

This is a bit about registers in this series of cache understanding.

In further articles, we will be discussing about the following

On-chip caches (SRAM)

- L1 Cache
- L2 Cache
- L3 Cache