Pipelining : Towards CPI = 1 or less than 1 Dream

“The principle of pipelining has emerged as a major architectural attribute of most present computer systems.

In particular, super machines such as the Texas Instruments TI ASC, Burroughs PEPE, IBM System/360 Models 91 and 195, Cray Research CRAY-1, CDC STAR-100, Amdahl 470 V/6, CDC 6600, and CDC 7600 have distinct pipeline processing capabilities, either in the form of internally pipelined instruction and arithmetic units or in the form of pipelined special purpose functional units .”

Pipeline Architecture *C. V. Ramamoorthy* *University of California, Berkeley, Berkeley and H.F. Li University of Illinois,* **Computing Surveys, Vo|. 9 No, 1, MARCH 1977**

Pipelining

• Pipelining is one form of imbedding parallelism or concurrency in a computer system.

• It refers to a segmentation of a computational process (say, an instruction) into several sub processes which are executed by dedicated autonomous units (facilities, pipelining segments).

•Successive processes (instructions) can be carried out in an overlapped mode analogous to an industrial assembly line.

• Technique of decomposing a repeated sequential process into sub processes, each of which can be executed efficiently on a special dedicated autonomous module that operates concurrently with the others.

Pipelining of Combinational Circuits

“CONVENTIONAL PIPELINING partitions the combinational logic into smaller chunks and inserts registers at the boundaries.

WAVE PIPELINING takes another approach, relying on the logic path being long enough and the data dispersion reasonably small, so the system can send multiple sets of data (waves) through the logic at a faster clock rate and without latching the data on the way.”

Automating Wave-Pipelined Circuit Design Woo Jin Kim
Sun Microsystems, Yong-Bin Kim Northeastern University,
IEEE Design & Test of Computers, November–December 2003

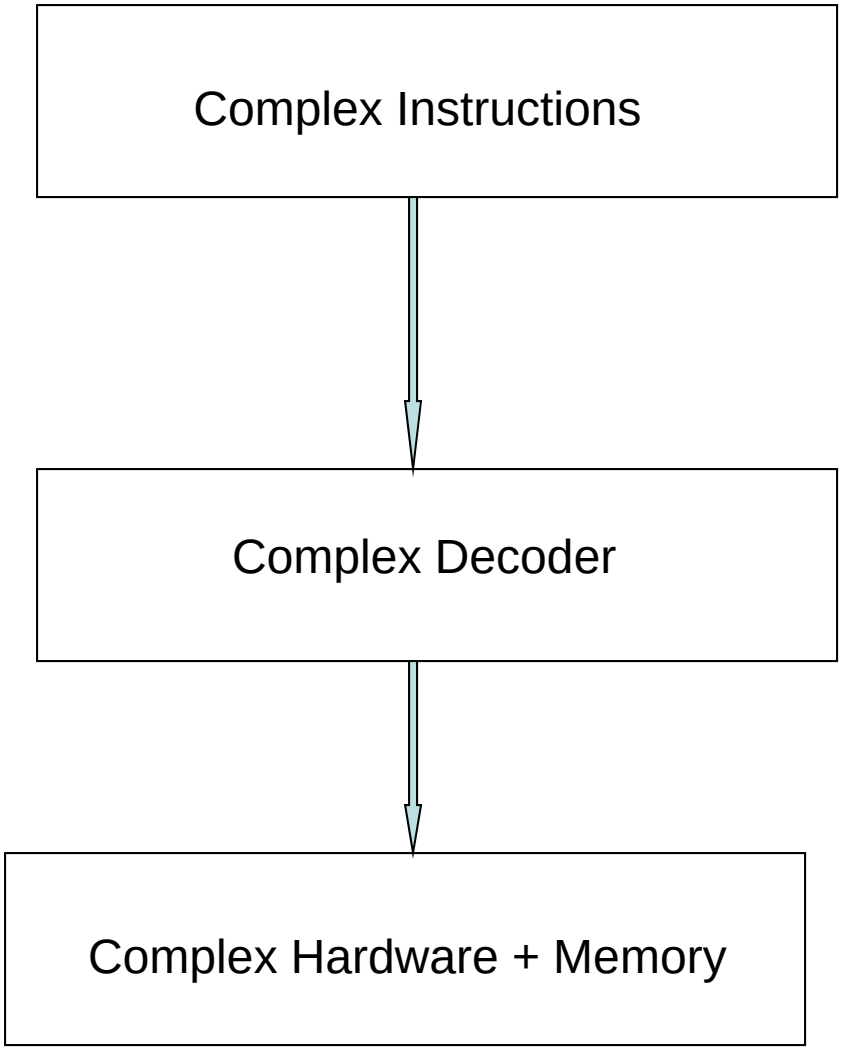
Ever Increasing SoC Design Complexity : Challenges

Large amount of information should be transported across the chip with negligible latency and error with high speed and high reliability

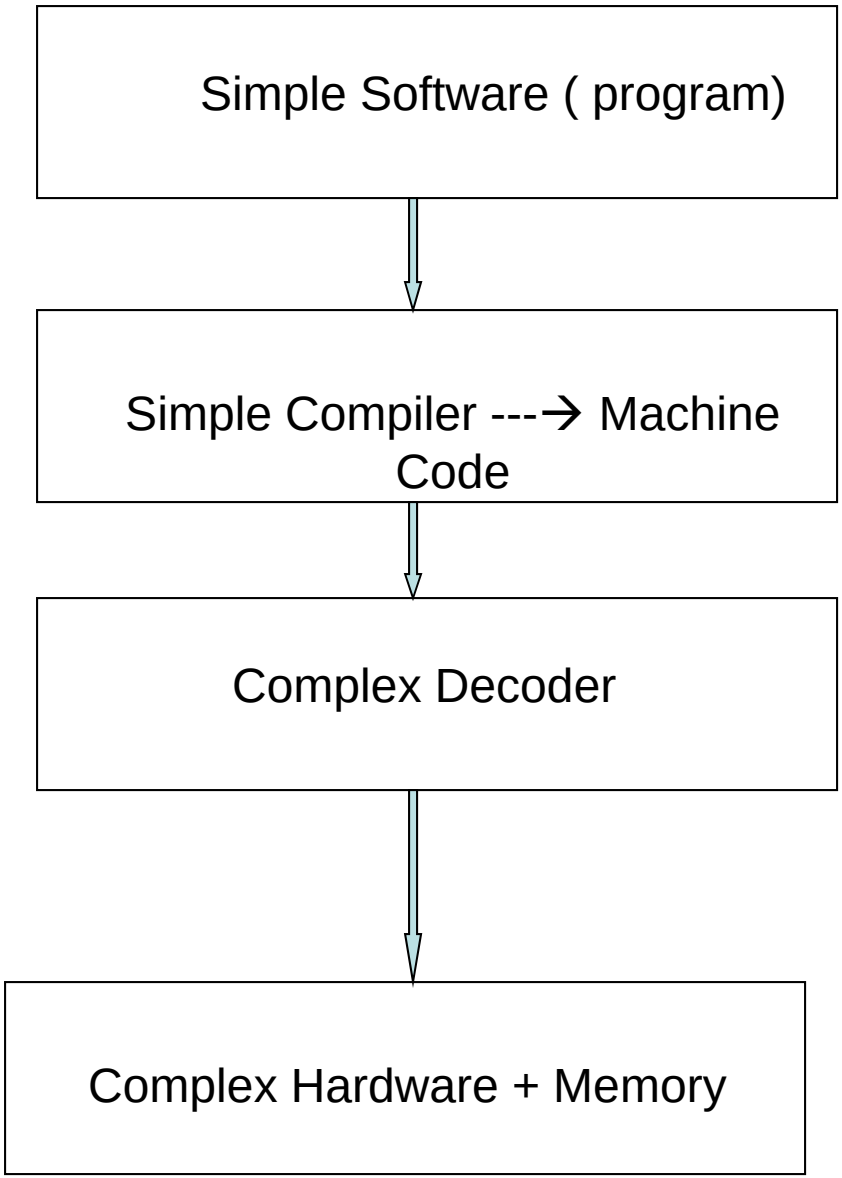
Design uncertainties in future SoC rise sharply due to coupling noises, process variations, and power delivery fluctuations.

The cost of maintaining nearly error-free transmission limits the attainable throughput rate.

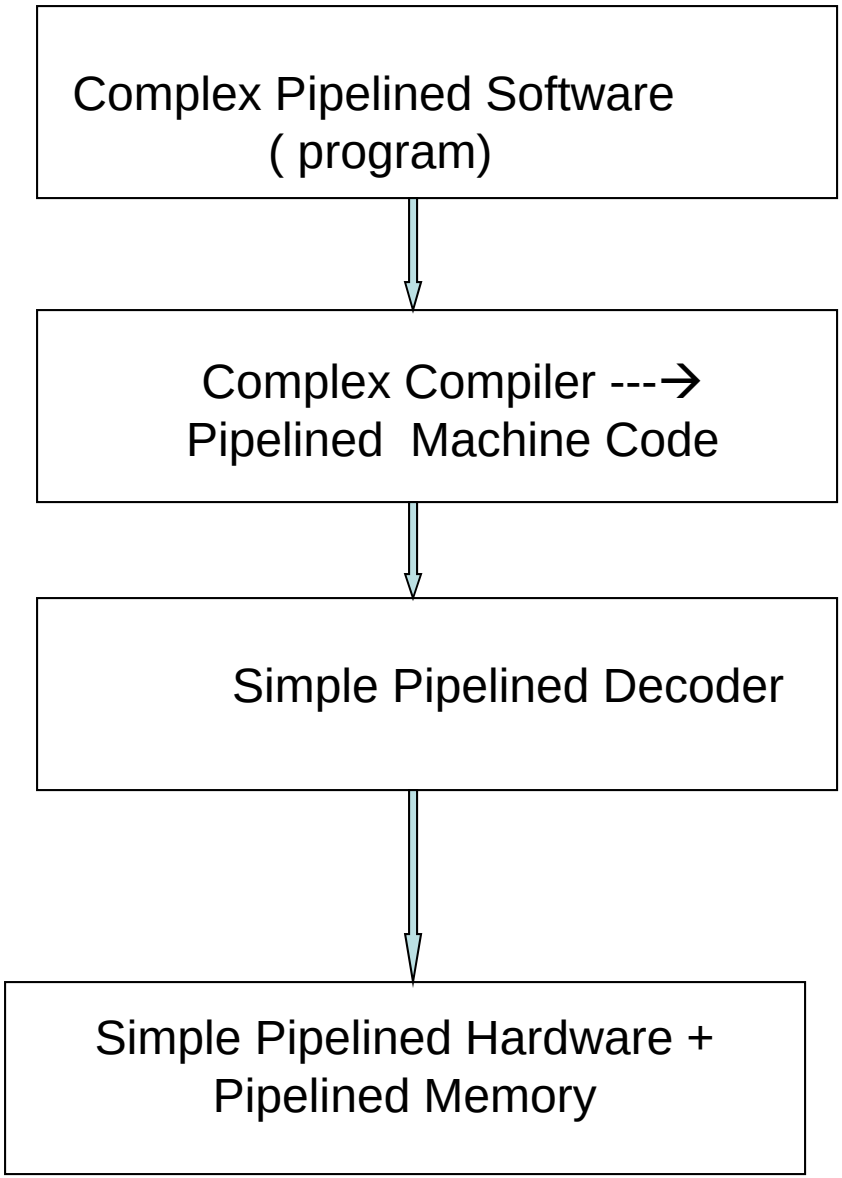
Embedded System Architecture : Power of Pipelining



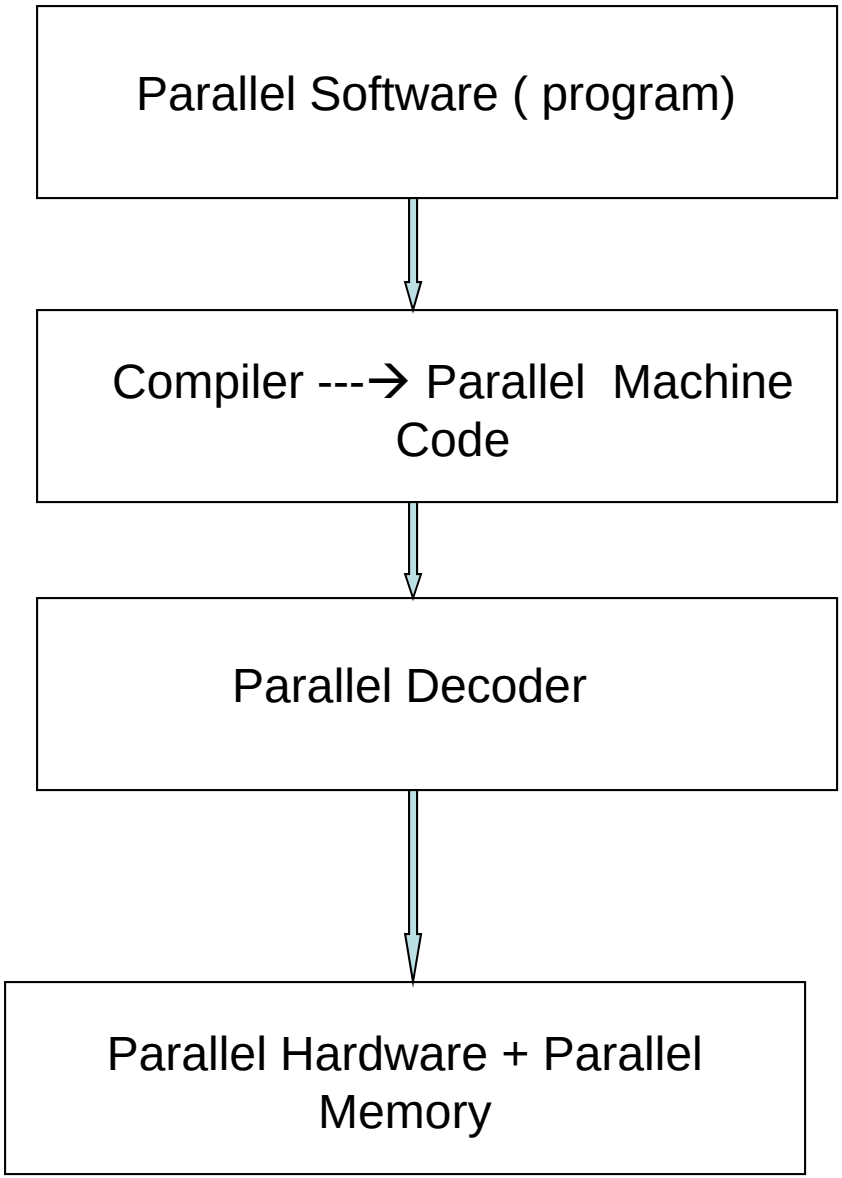
Embedded System Architecture



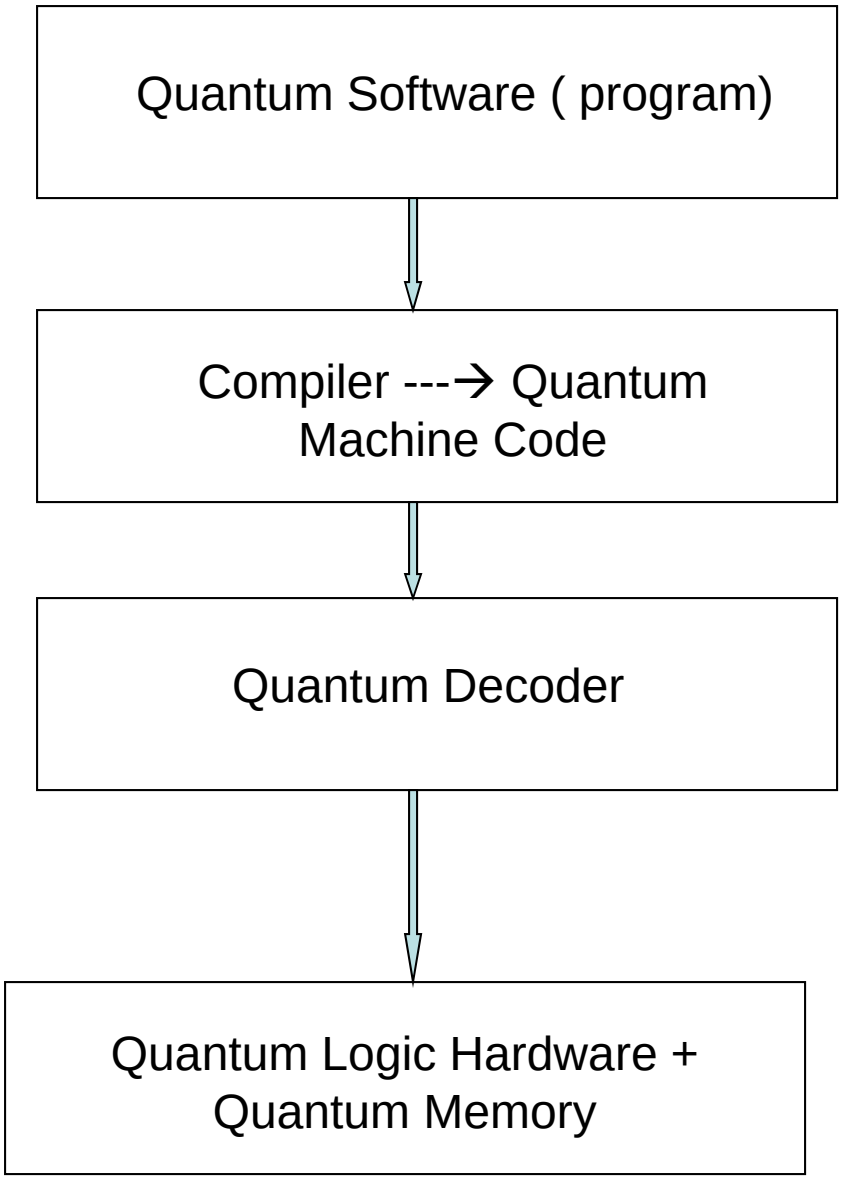
Embedded System Architecture



Embedded System Architecture



Embedded System Architecture



Pipelining

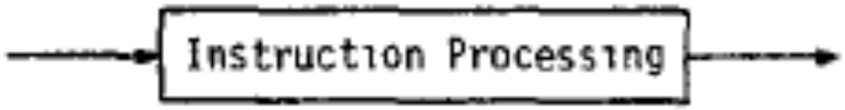


FIGURE 1a. Non-pipelined processor.

FIGURE 1b. Pipelined processor.

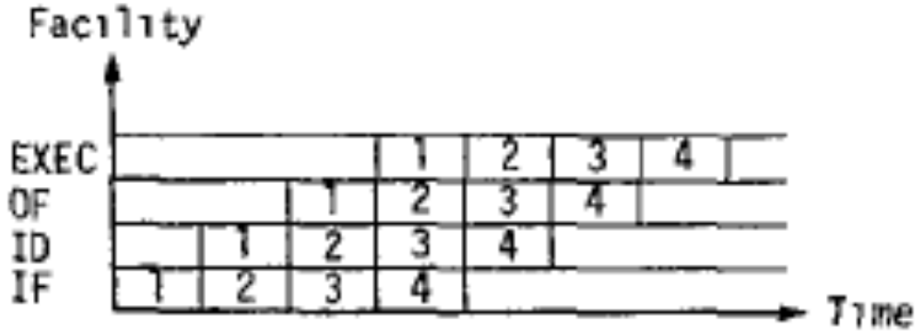


FIGURE 1c. Space-time diagram.

Pipelining

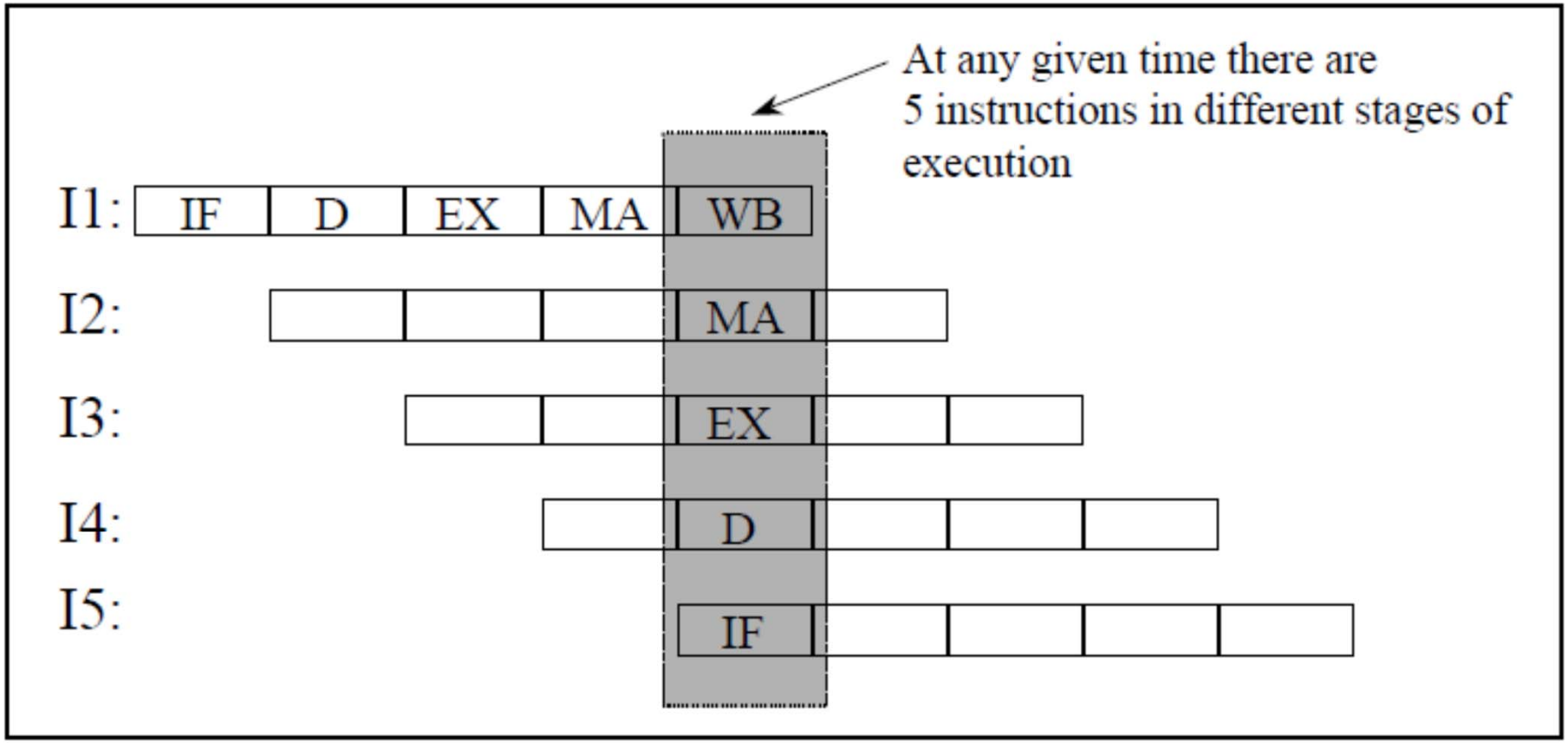


Fig. 1. *Typical five stage RISC pipeline*

A Pipeline Example

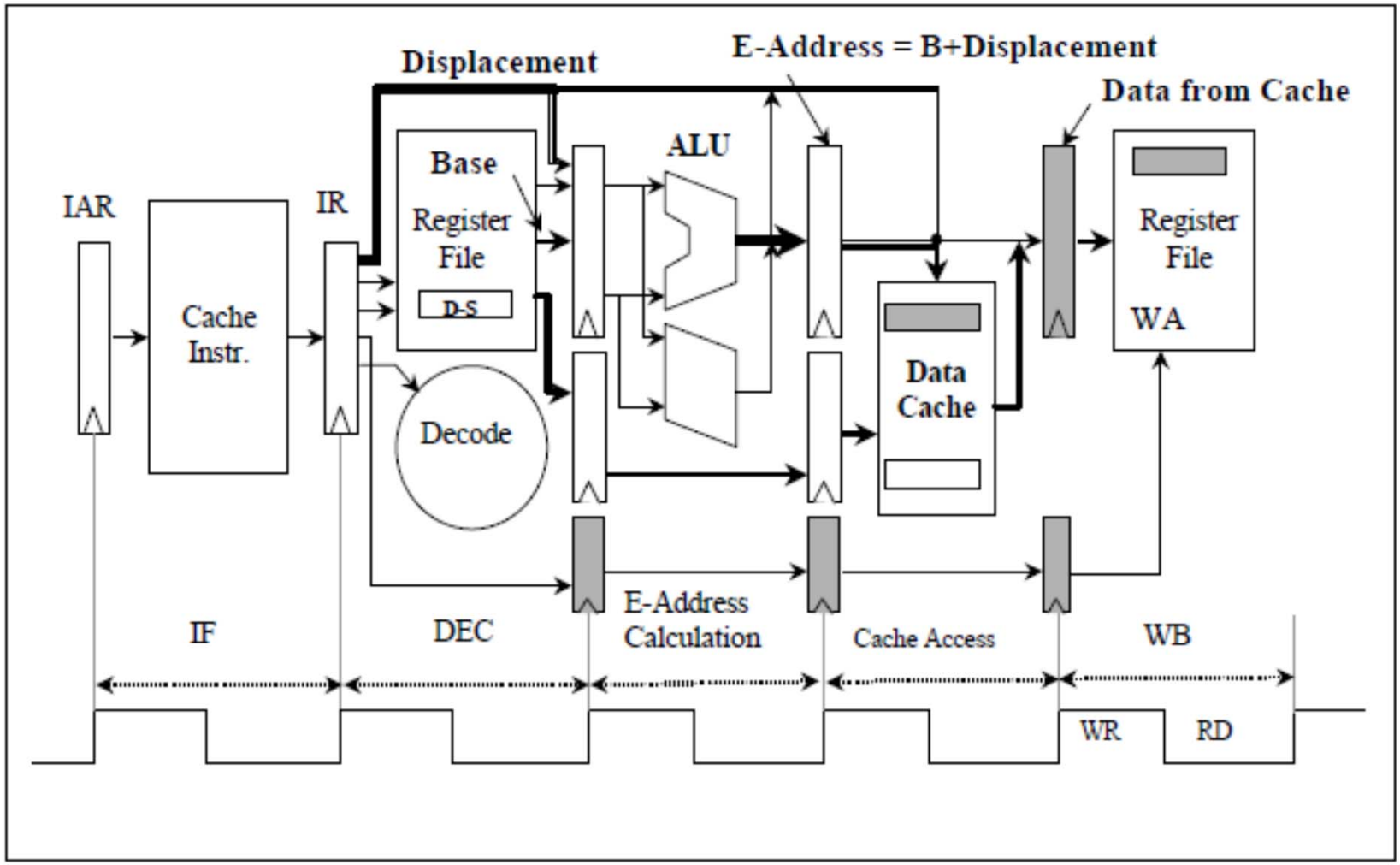


Fig. 3. *The Operation of Load/Store Pipeline*

Instruction Scheduling by Computer

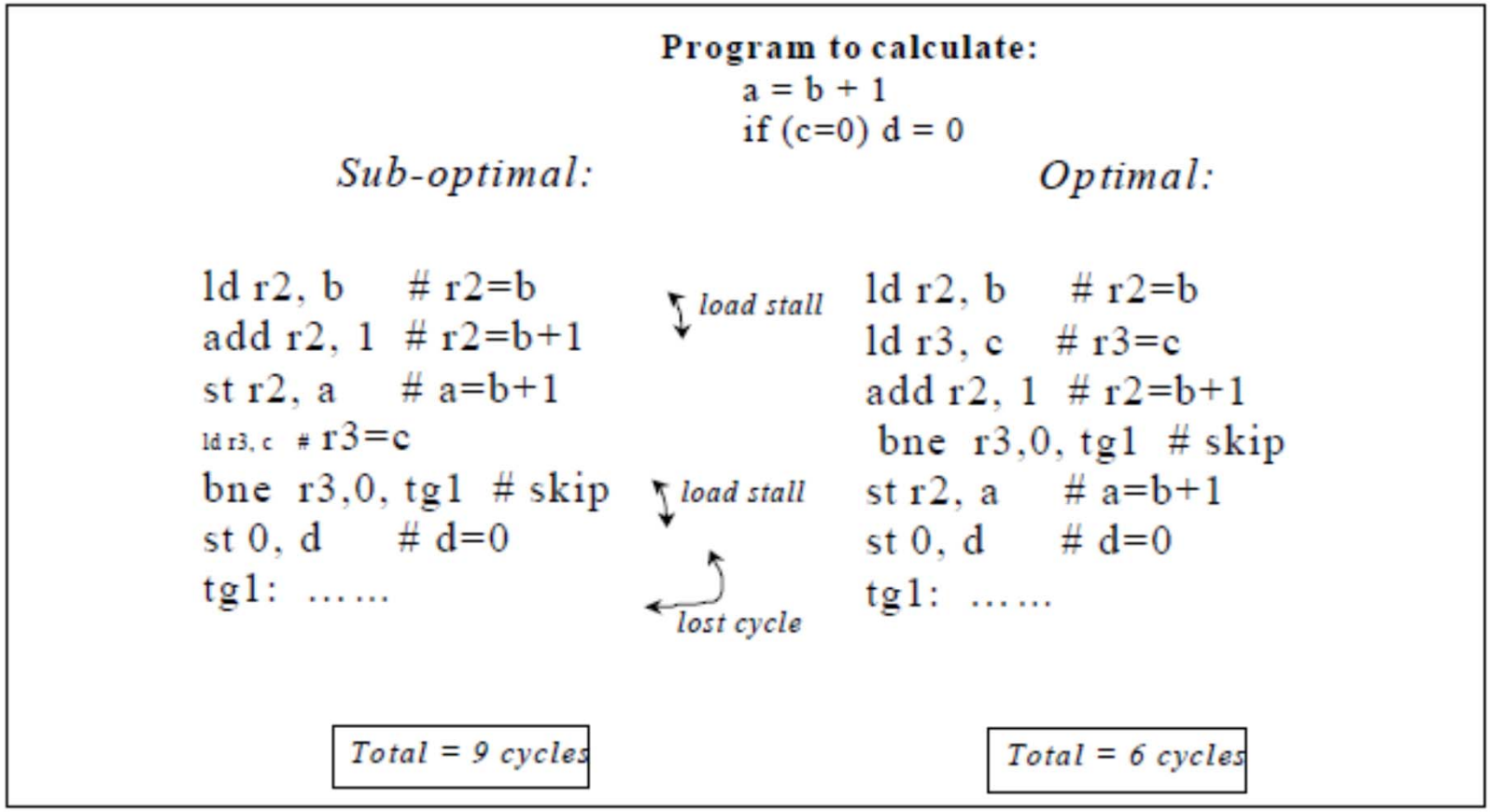


Fig. 7. *An Example of Instruction Scheduling by Compiler*

Pipeline Performance

Throughput rate: defined as the number of outputs (sometimes the number of instructions processed) per unit time.

Efficiency or Utilization factor: Percentage of busy (productive) periods with respect to a certain time span.

$$\text{efficiency of pipeline} = \frac{\text{total space-time span of tasks}}{\text{total space-time span of facilities}}.$$

$$\text{efficiency of pipeline} = \frac{\text{total weighted space-time span of } L \text{ tasks}}{\text{total weighted space-time span of } n \text{ facilities}}.$$

Throughput Consideration

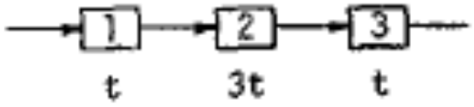


FIGURE 1f. Facility 2 is the bottleneck.

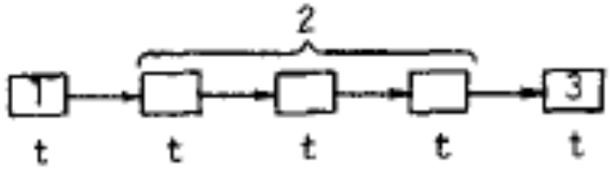


FIGURE 1g. Subdivision of facility 2.

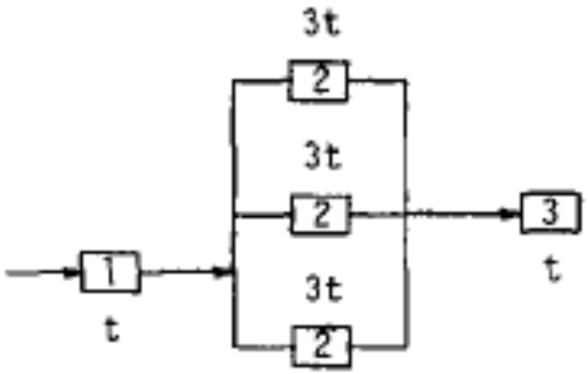


FIGURE 1h. Paralleling of facility 2.

"However, putting facilities in parallel creates more problems in distribution and synchronization of the tasks in the pipeline."

Pipeline Performance : Clock rate and maximum speed limitations

As data and control flow from one pipe segment to another, the propagation delay through each segment and the possible signal skews have to be carefully balanced to avoid any improper gating in a high speed situation.

In the maximum speed pipeline design, all segments have to be synchronized by the same clock for propagating the data through the pipe.”

Pipeline Performance : Bus Architecture

“Pipelining requires the concurrent processing of independent instructions though they can be in consecutive stages of execution. With dependent instructions their input and traversal through the pipe have to be paused until the dependency is resolved. Thus an efficient internal bus structure is needed to route the results to the requesting stations efficiently.”

Pipeline Performance : Branching

“Branching is more damaging to the pipeline performance than instruction dependency. When a conditional branch is encountered, one cannot tell which sequence of instructions will follow until the deciding result is available at the output. Therefore a conditional branch not only delays further execution but also affects the entire pipe starting from the instruction fetch segment. An incorrect branch of instructions and operands fetched may create a discontinuity of instruction supply.”

Pipeline Performance : Interrupt handling

“Interrupts disrupt the continuity of the instruction stream in a pipeline much as the conditional branches. When an interrupt occurs while instruction i is being executed, the interrupt should be serviced before any action is applied to instruction i + 1. This implies that either these two instructions are to be executed sequentially or sufficient information is set aside for the eventual recovery of instruction i + 1. The first course defeats the purpose of pipelining. The second approach is taken by some architectural designs when the cost of recovery is not overly substantial.”

Control Structure, Hazards, and Penalties

“The control structure of an overlapped or pipeline system is often overlooked. It plays such a significant role in characterizing the system under study that it determines the resulting operational efficiency.”

Two major control strategies

1. Streamline flow of instructions through the system, with one instruction (task) following another

• Completion ordering of the instructions is the same as their initiation ordering.

• Simple interlocks between two adjacent segments to allow the transfer of data control from one segment to the next.

• Interlock is necessary because the pipe is asynchronous, and some segments may have speeds different from others or variable depending on the control information.

• When a bottleneck appears dynamically at a segment, the input will be halted temporarily until the segment is free again.

Two major control strategies

2. A more flexible and powerful control structure

• The system can be viewed as fully asynchronous, so that completion ordering of the instructions need not be the same as their initiation ordering.

• When an instruction is held up because of some hazard condition, the next instruction may be allowed to go ahead.

•Such a scheme is desirable whenever the system has multiple (either physical or virtual) execution units or facilities running in parallel (besides the pipelining employed).

•In some cases the execution time of one instruction may be very different from that of another, and it is only natural to allow a subsequent short instruction to finish ahead of a preceding (but independent) long instruction.

Problems which need attention to implement control strategies

1) Read after write,
2) Write after write, and
3) Write after read.

These three hazards need separate detectors and resolvers. The location of a detector and the complexity of a resolver decide the penalty (time delay in initiation) that is incurred by the hazard.

Read after write

- Simultaneous execution (though in different phases) of several active instructions, the data needed by these instructions has to be guaranteed to be correct.

Example: For two "active" instructions, say i and j (j being an immediate successor of i), if i writes into a region and j needs to fetch some control or data from the same location in that region, a "read after write“ phenomenon occurs.

(The term "region“ is a flexible term that refers to any storage element, e.g., a register or main memory.

To synchronize i and j properly, j has to defer fetching that value until i has completed; otherwise, the wrong information (data or control) is used and the control scheme fails.

Write after write

If the instructions i and j write into the same region, even if i completes after j (this may occur when i is a long instruction or when something delays i), the resulting value stored in the region should reflect the result of instruction j, not i. So, to guarantee correct execution, the control structure has to resolve any such occurrence.

Write after Read

- This problem is less severe and rarely occurs except in some special cases.
- It involves the completion of a read before the next write to the same region takes place.
- If a read is initiated (to the memory), even if memory interference delays the actual read, a subsequent write to the same location will still follow the read.

- A potential situation where such a problem may need further control is when the read and write requests have separate request queues; requests on both queues then have to be synchronized for the write after read to guarantee that the write follows the read.

Advanced Risk Machines (ARM): A small Beginning

The history of ARM started in 1983, when a company named Acorn Computers was looking for a 16-bit microprocessor for their next desktop machine.

They quickly discovered that existing commercial microprocessors did not satisfy their requirements. These processors were slower than standard memory parts available at the time.

Therefore, Acorn engineers considered designing their own microprocessor. However, resources required for such a project were well beyond what the company could afford.

Advanced Risk Machines (ARM): A small Beginning

In early 80's, microprocessor architectures became so complex that it took many years even for large companies with significant expertise in the area to design a new processor.

Acorn decided to pick up the Berkeley RISC approach, and two years later, in 1985, they released their first 26-bit Acorn RISC Machine (ARM) processor, which also became the first commercial RISC processor in the world.

It used less than 25,000 transistors — a very small number even for 1985, but still performed comparably to or even outperformed the Intel 80286 processor that came out at about the same time.

Advanced Risk Machines (ARM): A small Beginning

At its inception ARM stood for Acorn RISC Machine. The first ARM reliant systems include the Acorn: BBC Micro, Masters, and the Archimedes.

During this early period they were used mostly for British educational systems, and therefore, were not widely available or known outside England.

In 1987 the ARM became the first commercial RISC processor.

ARM is a a 32-bit RISC processor architecture.

Joint Venture by Apple and Acorn : Advanced Risk Machines

In 1990 Apple made a strategic decision to use an ARM processor in their Newton PDA.

The third version of the ARM architecture was developed by this company and featured 32-bit addressing, MMU support and 64-bit multiply accumulate instructions. It was implemented in ARM 6 and ARM 7 cores.

The release of these processors and the Apple Newton PDA in 1992 marked ARM’s move to the embedded market.

Advanced Risk Machines : Thumb Instruction

The 4th generation of ARM cores came out in 1996.

The main innovation in this version of the architecture was support for Thumb 16-bit compressed instruction set. Thumb code takes 40% less space compared to regular 32-bit ARM code but is slightly less efficient.

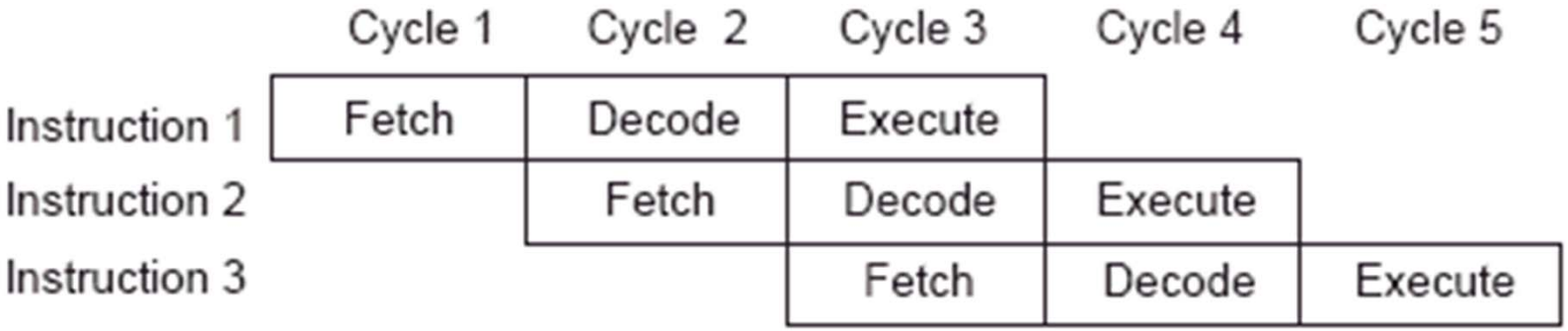
The most prominent representative of the 4th generation of ARM's is the ARM7TDMI core, which still remains the most popular ARM product

It is used in most Apple iPod players, including the video iPod.

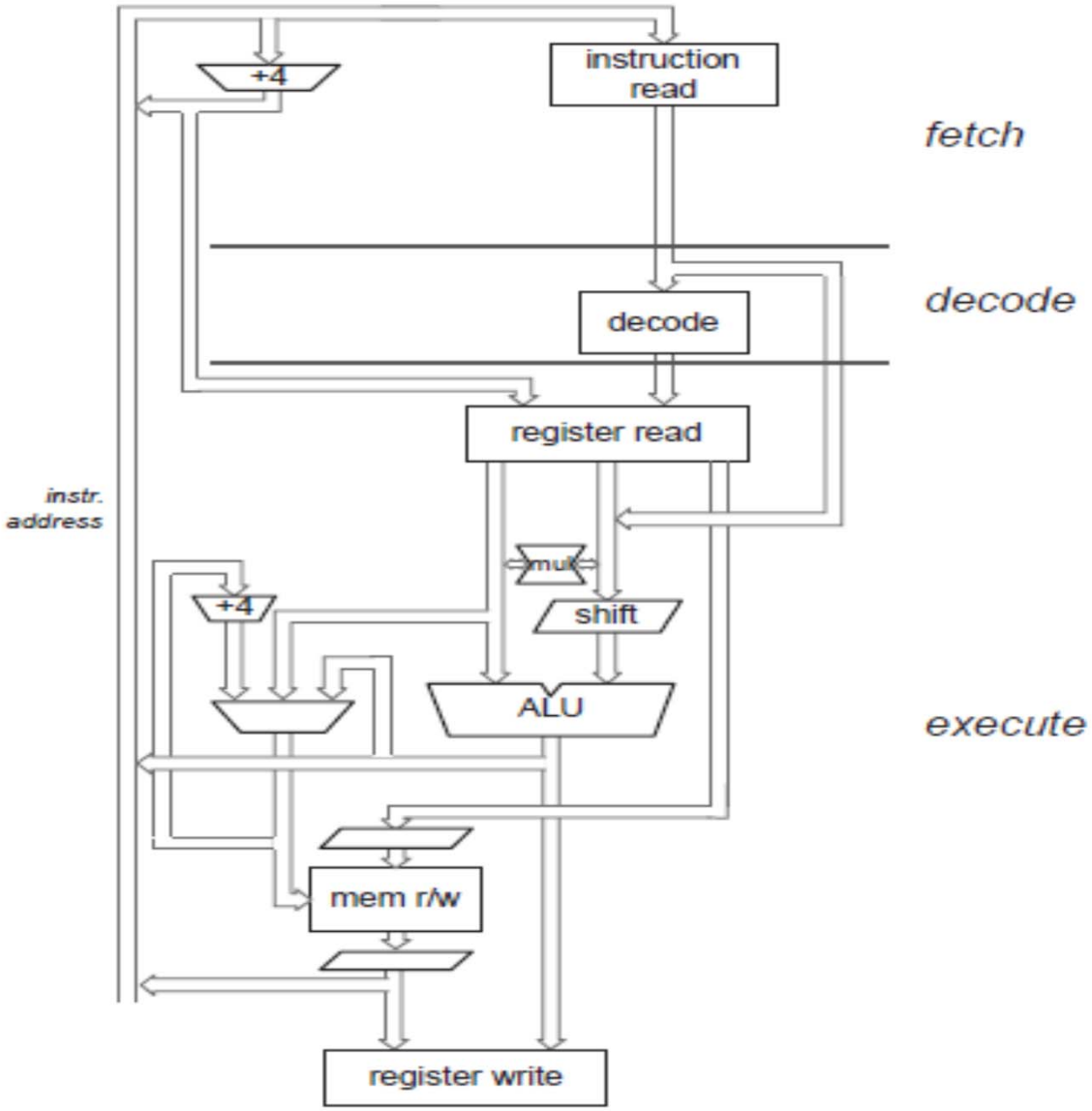
ARM7TDMI is based on essentially the same 3-stage pipeline as the very first ARM designed in 1985, and contains only 30000 transistors.

ARM7TDMI Pipelining

- Fetch – instruction is fetched from memory and placed in pipeline;
- Decode – instruction is decoded and data-path signals prepared for next cycle;
- Execute – instruction from prepared data-path reads from registry bank, shifts operand to ALU and writes generated result to dominant register.

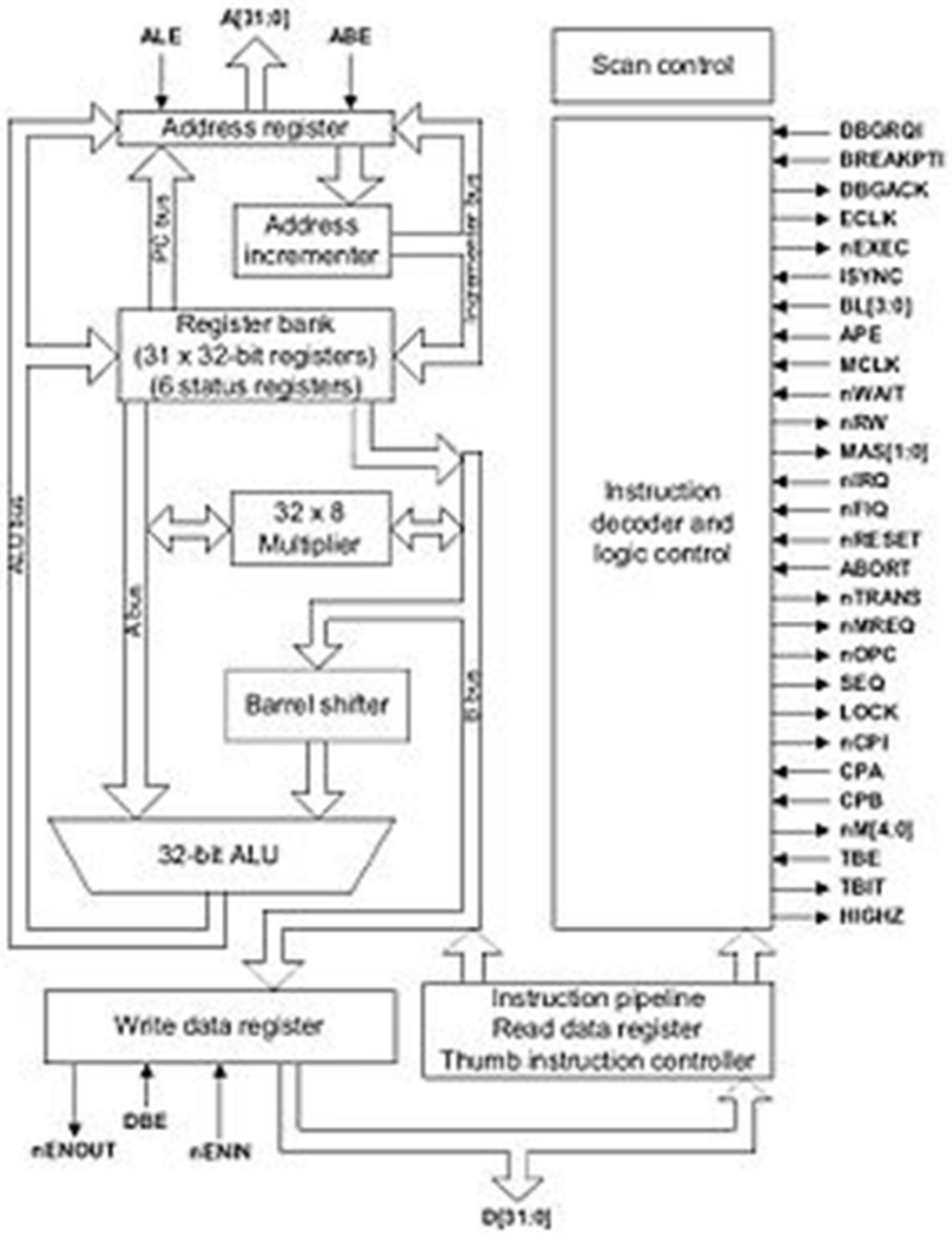


ARM7TDMI Pipelining



(a) ARM7TDMI

ARM7TDMI : ARM Thumb Instruction, Debugger, Multiplier, ICE



Advanced Risk Machines (ARM): Sales pitch

"The ARM architecture has

The best MIPS to Watts ratio in industry

The best MIPS to $ ratio in the industry

The smallest CPU die size

All the necessary computing capability coupled with low power consumption of which a highly flexible and customizable set of processors are available with options to choose from, all at a low cost."

Advanced Risk Machines (ARM): Business Model

Business Model is based on licensing the ARM architecture to companies that want to manufacture ARM-based CPU's or system-on-a-chip products.

Two main types of licenses are the Implementation license and the Architecture license.

The Implementation license provides complete information required to design and manufacture integrated circuits containing an ARM processor core.

ARM licenses two types of cores: soft cores and hard cores. A hard core is optimised for a specific manufacturing process, while a soft core can be used in any process but is less optimised.

The architecture license enables the licensee to develop their own processors compliant with the ARM ISA.

ARM the most popular embedded architecture today.

First, ARM cores are very simple compared to most other general-purpose processors, which means that they can be manufactured using a comparatively small number of transistors, leaving plenty of space on the chip for application-specific macrocells.

A typical ARM chip can contain several peripheral controllers, a digital signal processor, and some amount of on-chip memory, along with an ARM core.

Second, both ARM ISA and pipeline design are aimed at minimising energy consumption — a critical requirement in mobile embedded systems.

ARM Popularity

Third, the ARM architecture is highly modular: the only mandatory component of an ARM processor is the integer pipeline; all other components, including caches, MMU, floating point and other co-processors are optional, which gives a lot of flexibility in building application-specific ARM-based processors.

Finally, while being small and low-power, ARM processors provide high performance for embedded applications. For example, the PXA255 XScale processor running at 400MHz provides performance comparable to Pentium 2 at 300MHz, while using fifty times less energy.

Functions of Memory Management Unit in ARM

It translates virtual addresses into physical addresses,

It controls memory access permissions.

The MMU hardware required to perform these functions consists of a Translation Look aside Buffer (TLB), access control logic, and translation table logic.

The MMU supports memory accesses based on Sections or Pages. Sections are comprised of 1MB blocks of memory.

The MMU also supports the concept of domains - areas of memory that can be defined to possess individual access rights. The Domain Access Control Register is used to specify access rights for up to 16 separate domains.

ARM7TDMI Pipelining

- Pipelining is linear and implemented in hardware level.
- A simple data processing processor executes one instruction in single clock cycle while individual instruction takes three clock cycles.
- When program structure has branches then pipeline faces difficulties, because it cannot predict which command will be next.
- In this case pipeline flushes and has to be refilled which means execution speed drops from 1 instruction to 3 clock cycles.
- ARM instructions achieve optimal performance by making PC (Program Counter) calculated **8 bytes ahead** of current instruction.

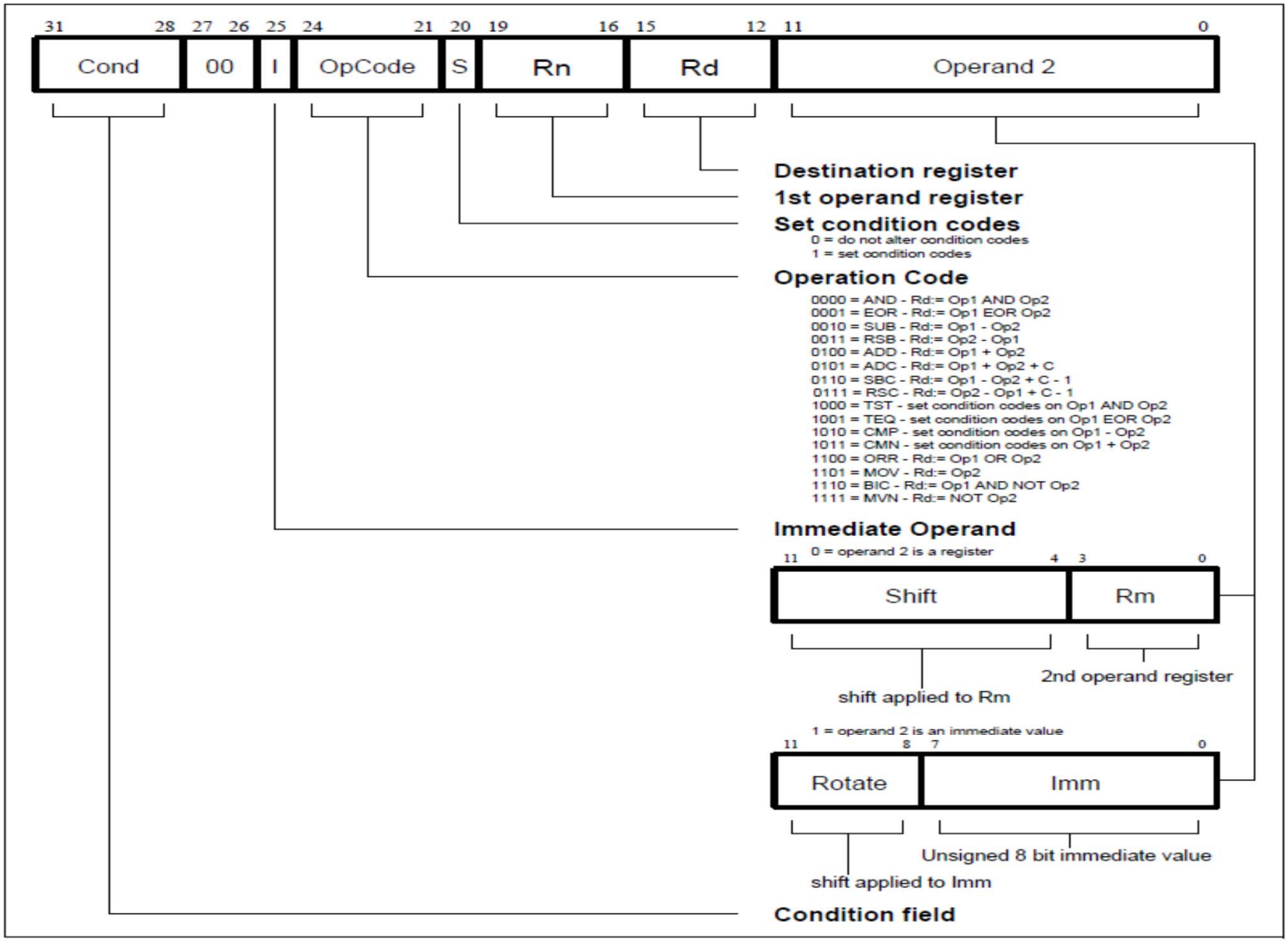
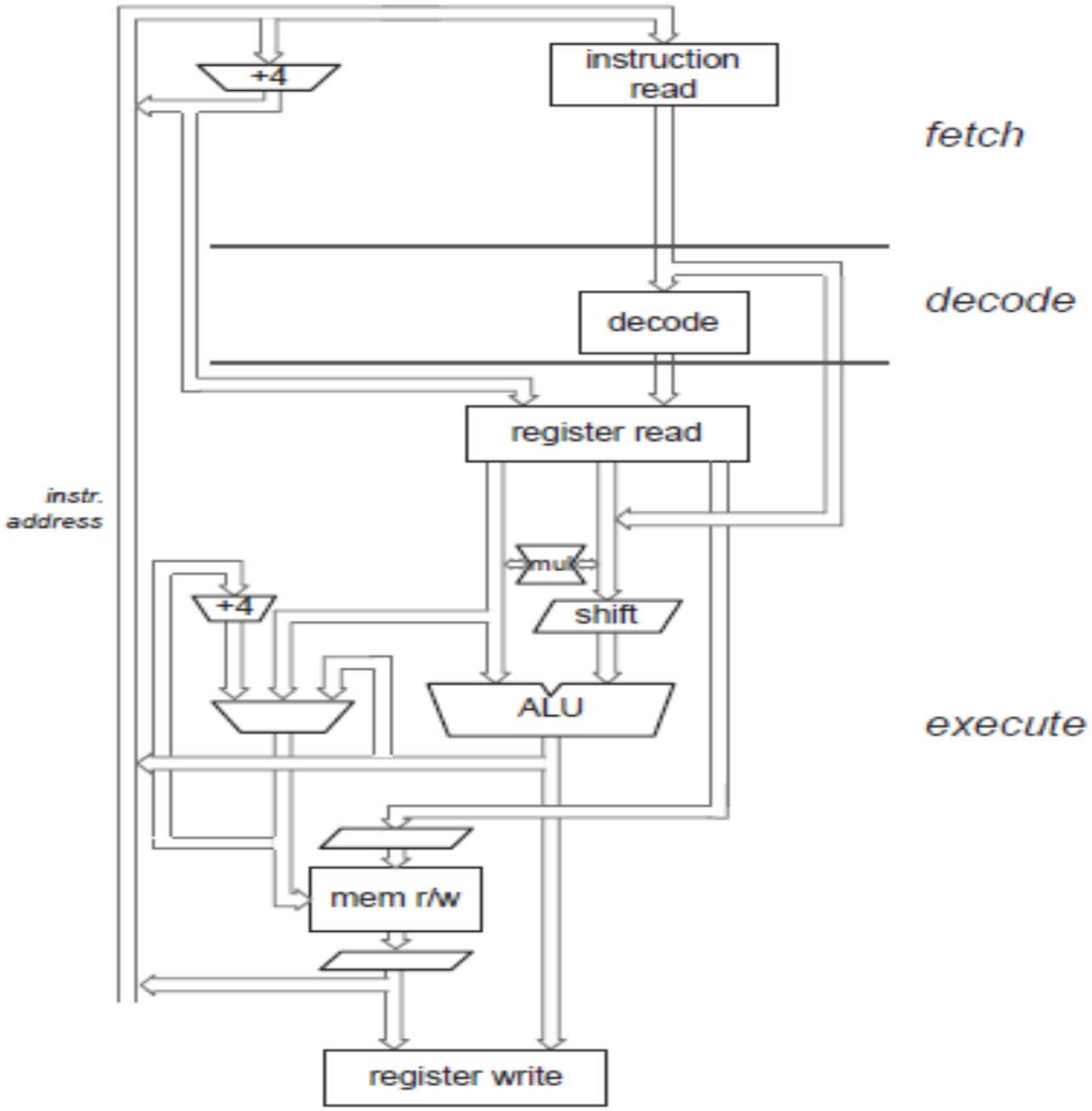


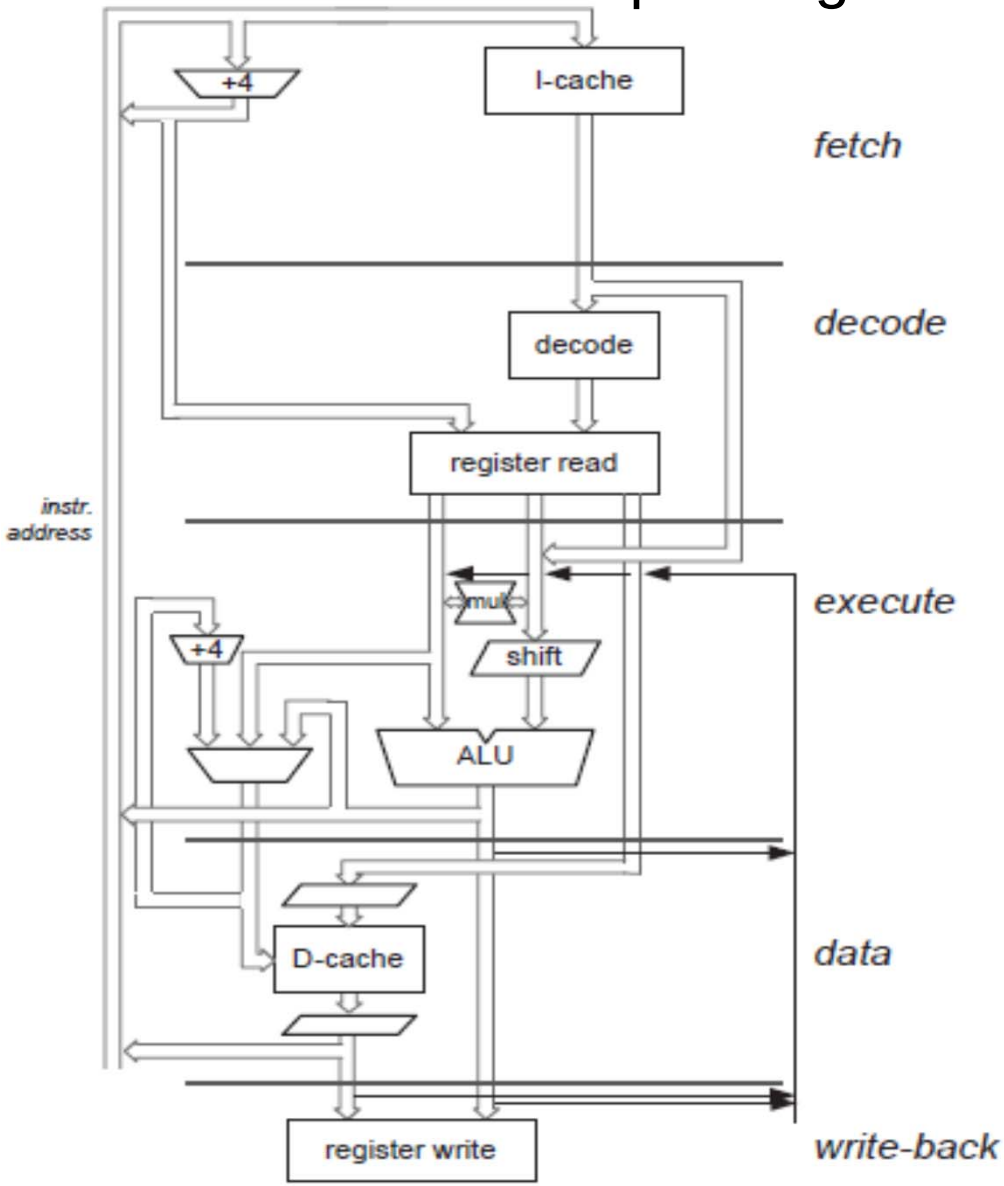
Figure 4-4: Data processing instructions

ARM7TDMI Pipelining



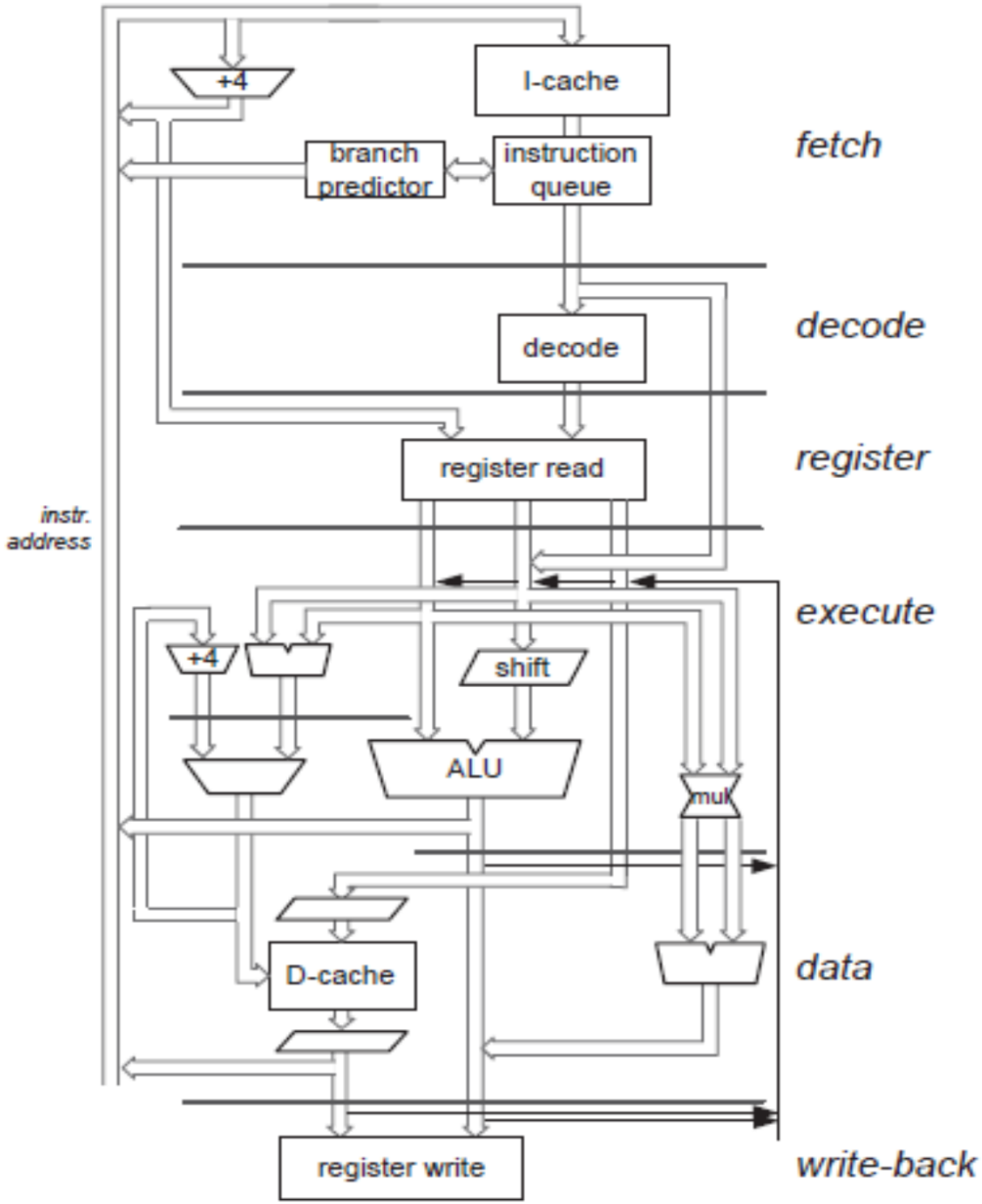
(a) ARM7TDMI

ARM9TDMI Pipelining



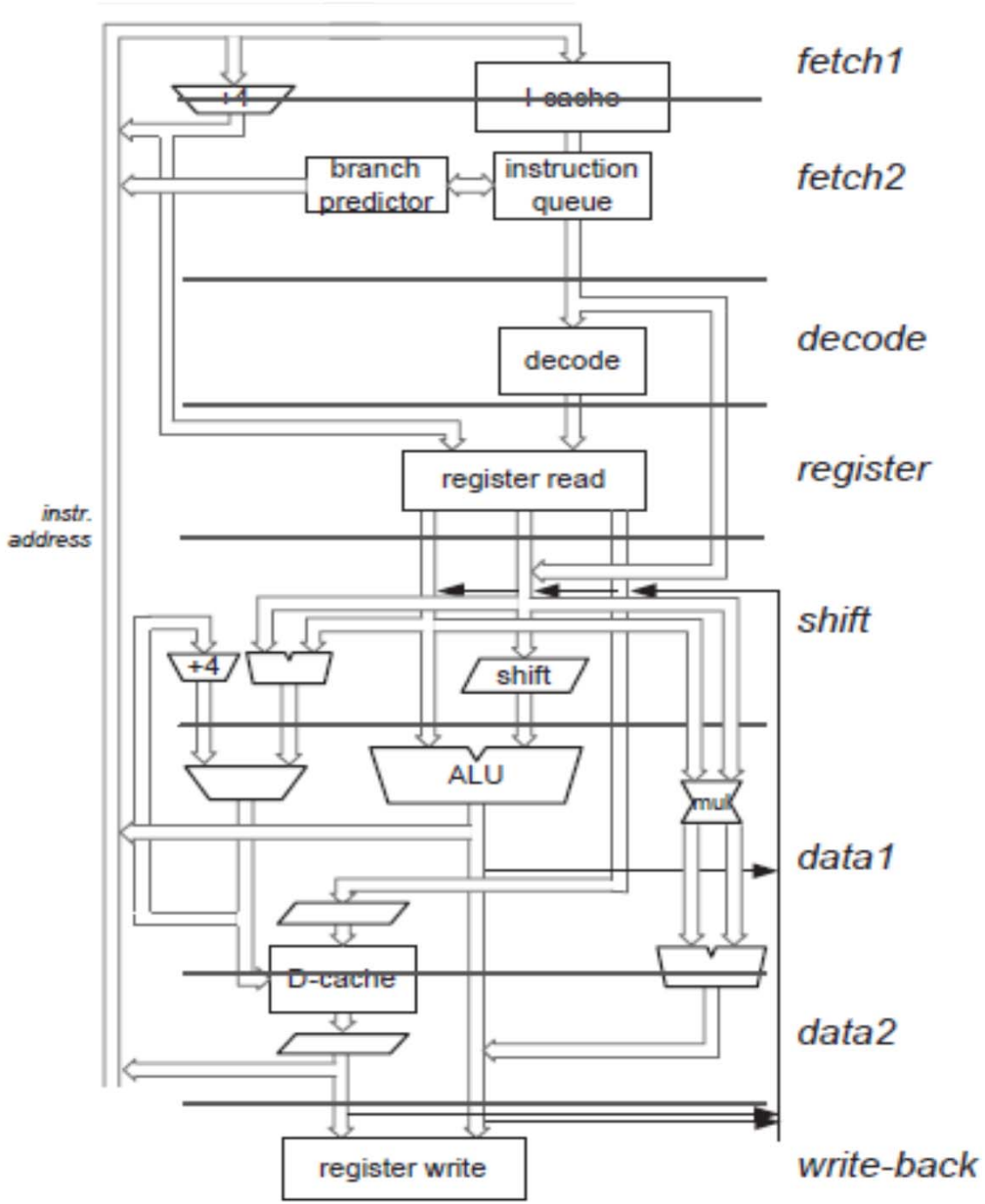
(b) ARM9TDMI

ARM10TDMI Pipelining



(c) ARM10TDMI

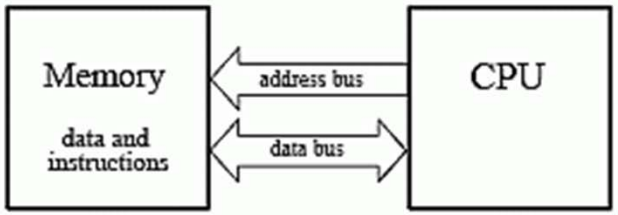
ARM11 Pipelining



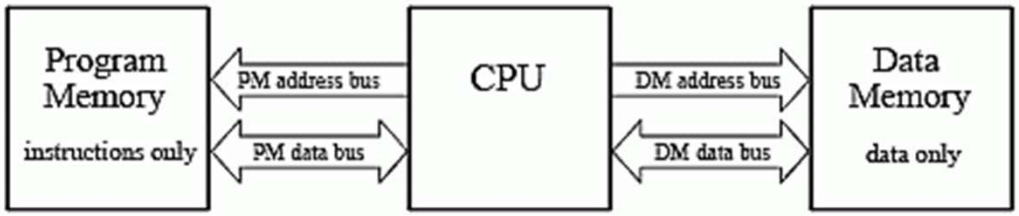
(d) ARM11

Who Wins in the Future?

a. Von Neumann Architecture (*single memory*)



b. Harvard Architecture (*dual memory*)



c. Super Harvard Architecture (*dual memory, instruction cache, I/O controller*)

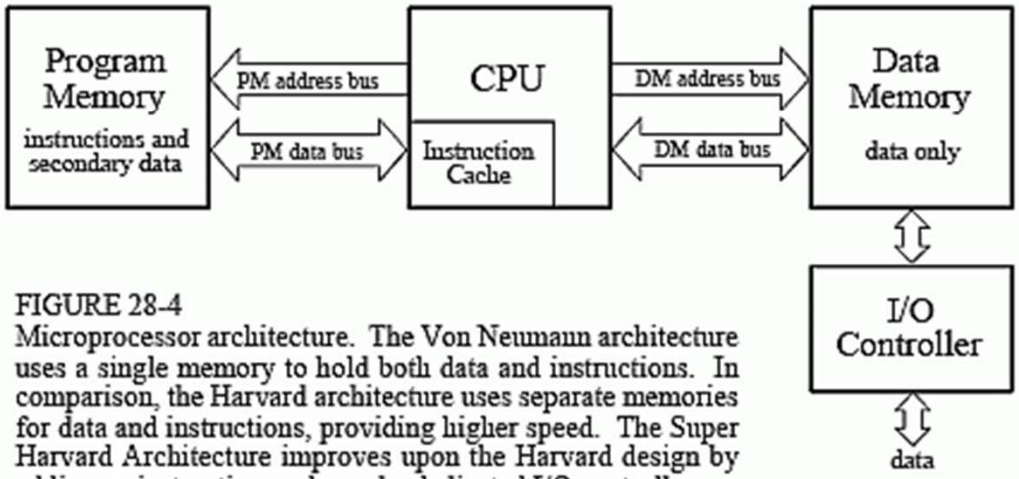


FIGURE 28-4
Microprocessor architecture. The Von Neumann architecture uses a single memory to hold both data and instructions. In comparison, the Harvard architecture uses separate memories for data and instructions, providing higher speed. The Super Harvard Architecture improves upon the Harvard design by adding an instruction cache and a dedicated I/O controller.

References

1. **Pipeline Architecture** ***C. V. Ramamoorthy*** *University of California, Berkeley, Berkeley and H.F. Li University of Illinois*