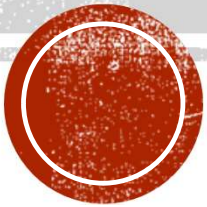
PROCESSORS AND MEMORY HIERARCHY

Module 2



OVERVIEW

- This chapter presents modern processor technology and the supporting memory hierarchy.
- We begin with a study of instruction-set architectures including CISC and RISC, and we consider typical superscalar, VLIW, superpipelined, and vector processors.
- The third section covers memory hierarchy and capacity planning and final section introduces virtual memory, address translation and page replacement methods.

4.1 ADVANCED PROCESSOR TECHNOLOGY

Architectural families of modern computers are

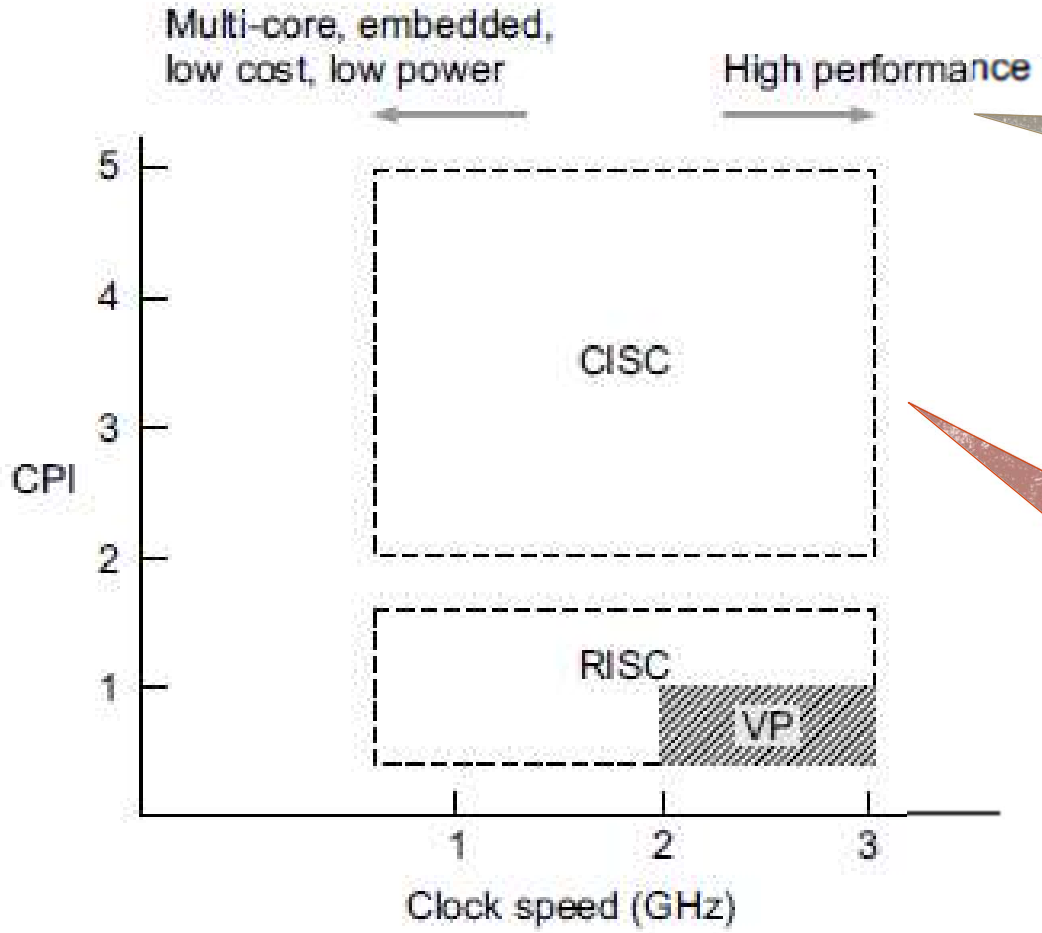
- CISC
- RISC
- Superscalar
- VLIW
- Super pipelined
- Vector processors
- Symbolic processors

Scalar and vector processors are numerical computation.

Symbolic processors have been developed for AI applications.

4.1.1 DESIGN SPACE OF PROCESSORS

- Various processor families can be mapped onto a coordinated space of **clock rate** versus **cycles per instruction(CPI).**
  - Clock speed is the amount of cycles the CPU can handle in one second and CPI means the amount of cycles it takes for the CPU to complete the instruction.
- As implementation technology evolves rapidly, the clock rates of various processors are gradually moving from low to higher speeds toward the right of the design space.
- Manufacturers are trying to lower the CPI rate using hardware and software approaches.



Under both CISC and RISC categories, products designed for multi-core chips, embedded applications, or for low cost and/or low power consumption, tend to have lower clock speeds. High performance processors must necessarily be designed to operate at high clock speeds.

The CPI of different CISC instructions varies from 1 to 20. Therefore, CISC processors are at the upper part of the design space.

Fig. 4.1 CPI versus processor clock speed of major categories of processors

CISC processors:

Intel i486,M68040,VAX/8600,IBM 390.

Clock rate: 33 to 50 MHz.

CPI:varies from 1 to 20 cycles.

Microprogrammed control.

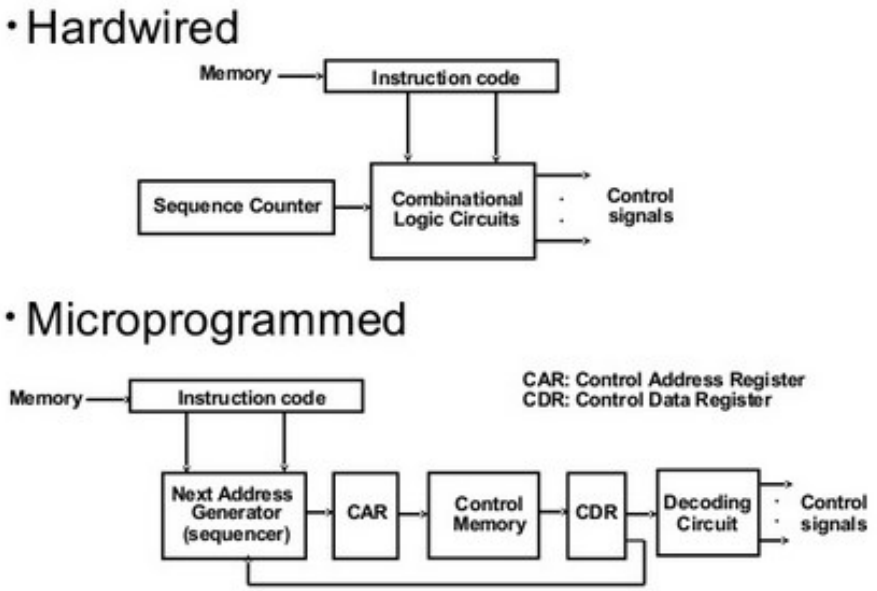
RISC processors:

Intel i860,SPARC,MIPS R3000,IBM RS/6000.

Clock rate: 20 to 120 MHz.

CPI:1 to 2 cycles.

Hardwired control.



Super scalar processors:

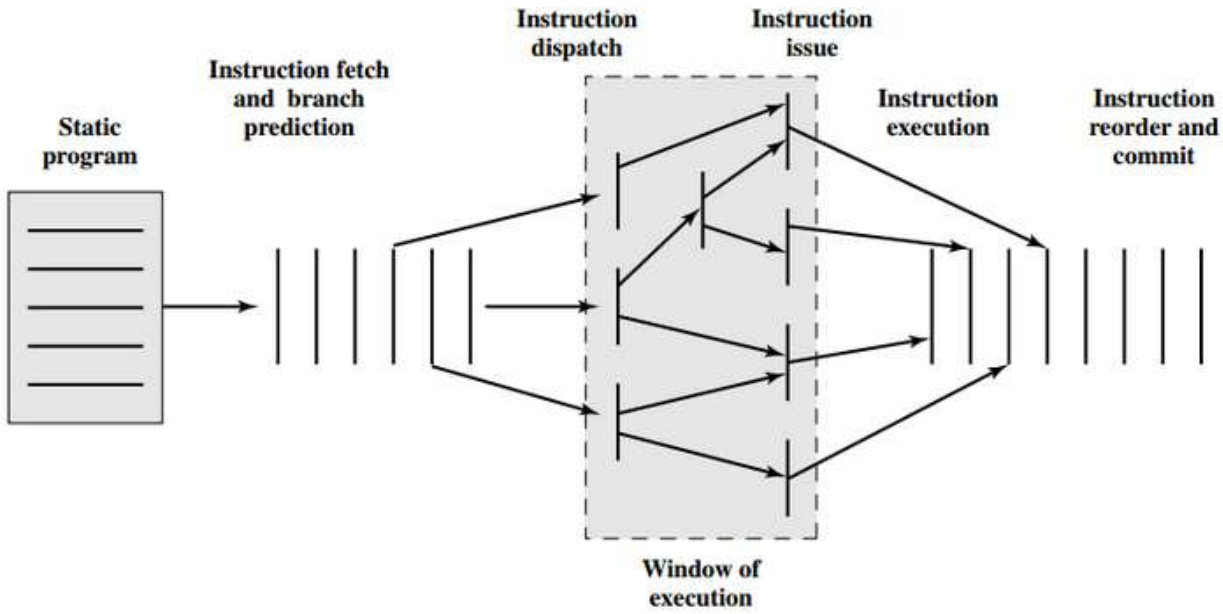
Intel i960CA,IBM RS/6000,DEC 21064.

Multiple instructions are issued simultaneously during each cycle.

Clock rate: 20 to 120 MHz.

CPI: .2 to .5 cycles.

- Subclass of RISC processors



Very long instruction word(VLIW):

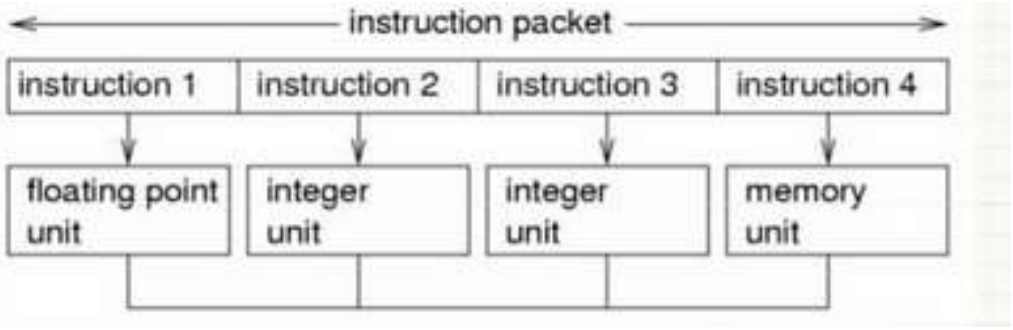
Uses more functional units than superscalar processor.

Clock rate: 5 to 50 MHz.

CPI: .1 to .2 cycles.

Uses very long instructions (256 to 1024 bits per instruction.)

Implemented with micro programmed control.



Super pipelined processors:

Uses multiphase clocks with a much increased clock rate.

Clock rate: 100 to 500 MHz.

CPI: 1 to 5 cycles.

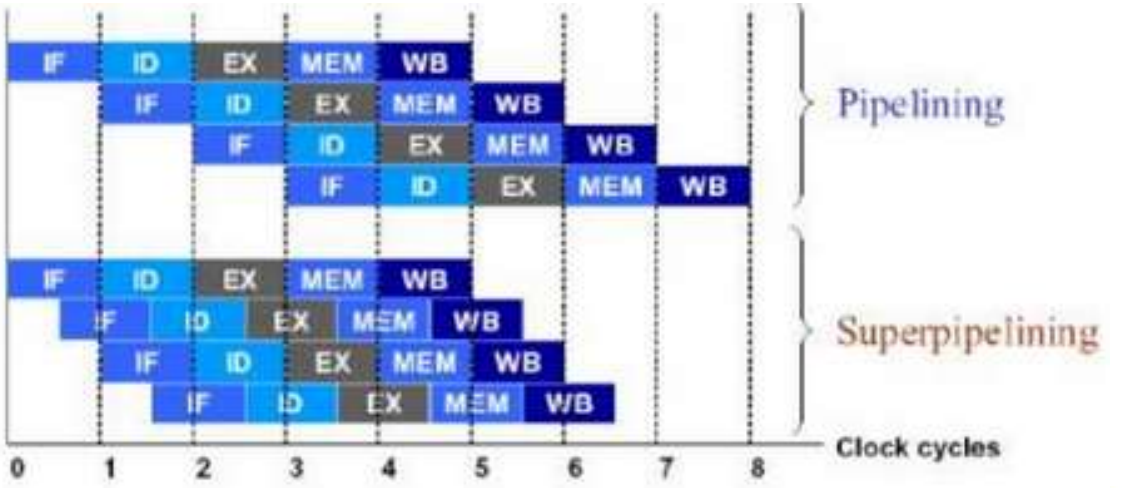
Vector supercomputers:

Uses multiple functional units for concurrent scalar and vector operations.

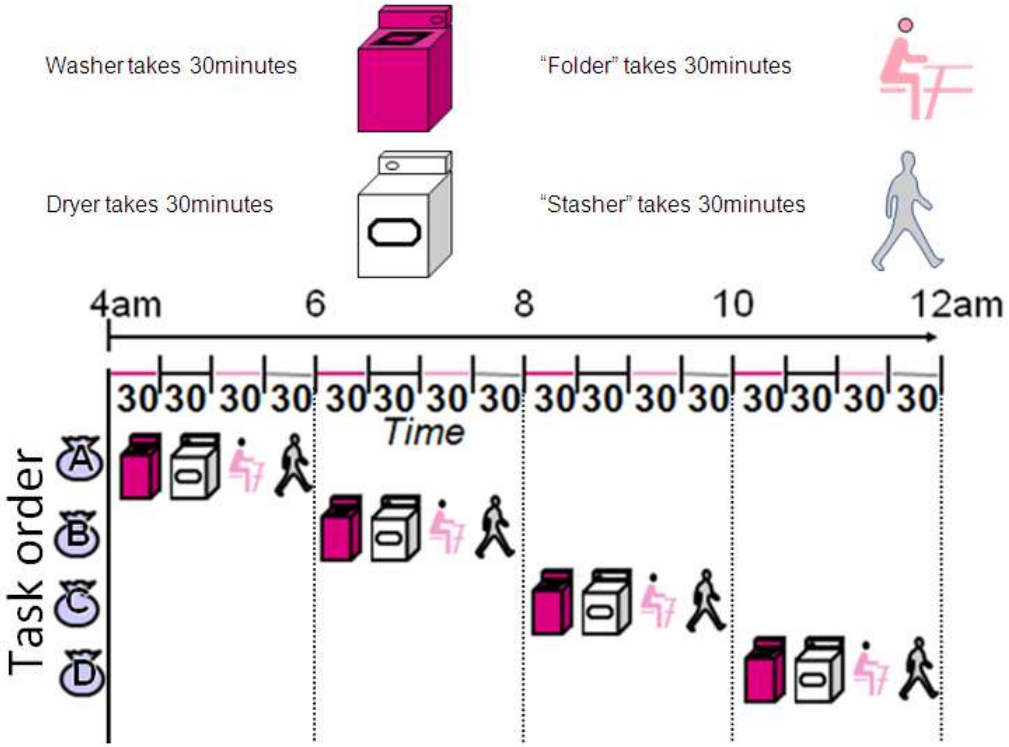
Processors are super pipelined.

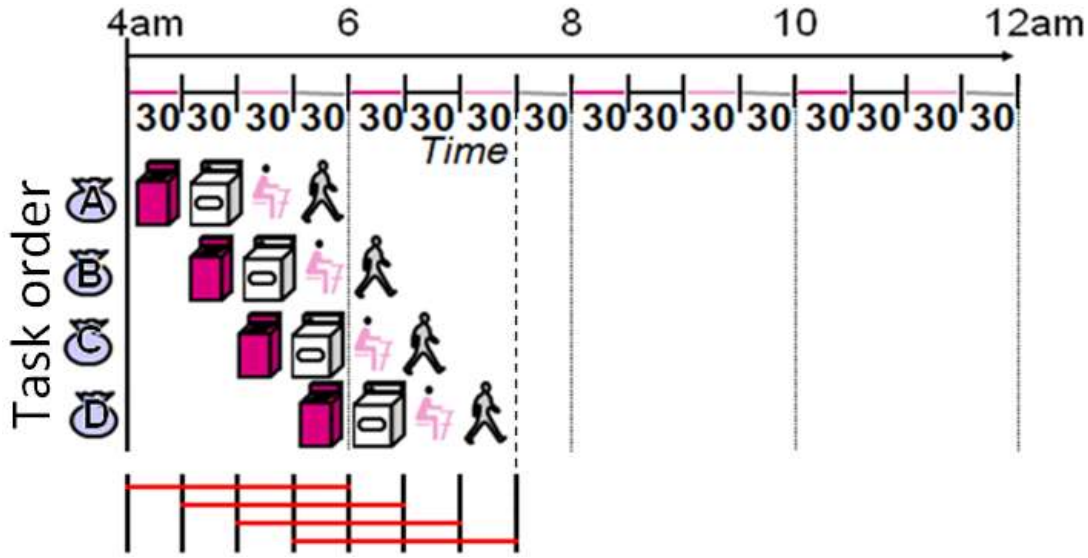
Very high clock rate:100 to 1000 MHz.

Very low CPI:.1 to .2 cycles.



WHAT IS PIPELINING?

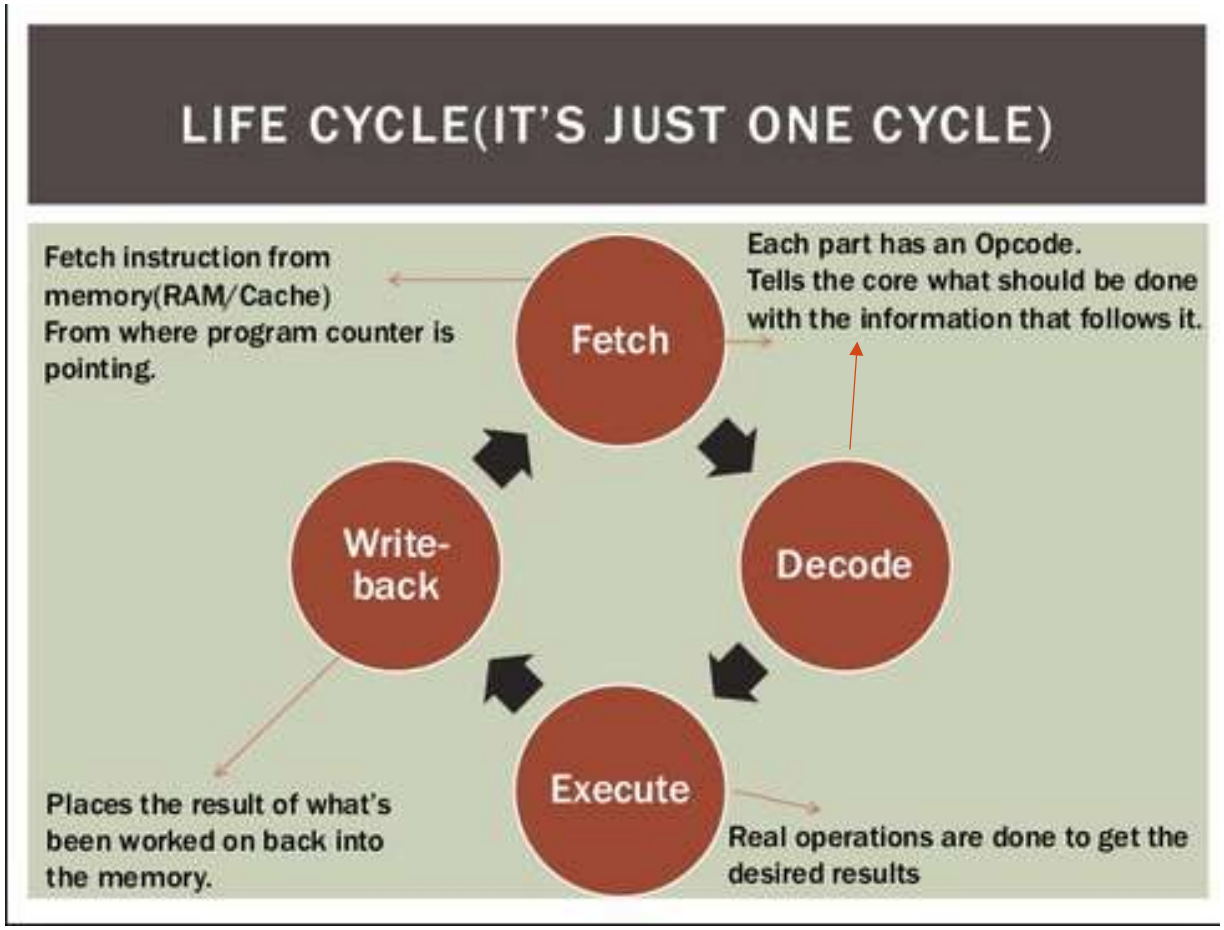




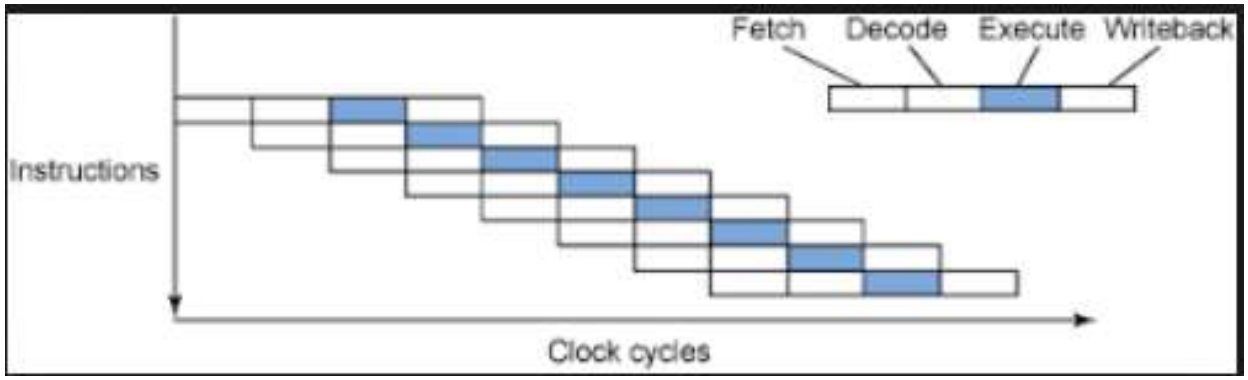
INSTRUCTION PIPELINES

The execution cycle of a typical instruction includes four phases.

- Fetch
- Decode
- Execute
- Write-back



Pipelining is an implementation technique where multiple instructions are overlapped in execution. The computer pipeline is divided in **stages**. Each stage completes a part of an instruction in parallel. The stages are connected one to the next to form a pipe - instructions enter at one end, progress through the stages, and exit at the other end.



Pipelining does not decrease the time for individual instruction execution.
Instead, it increases instruction throughput.

Pipeline cycle: It is defined as the time required for each phase to complete its operation assuming equal delay in all phases.

Instruction pipeline cycle: the clock period of the instruction pipeline.

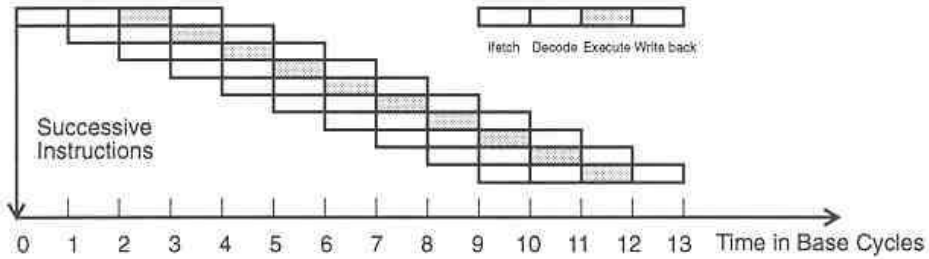
Instruction issue latency: the time (in cycles) required between the issuing of two adjacent instructions.

Instruction issue rate: the number of instructions issued per cycle.

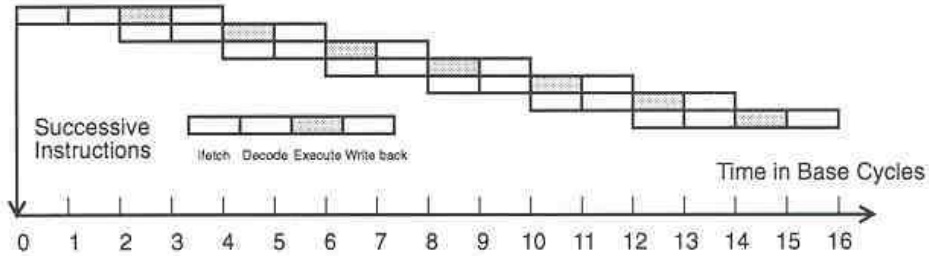
Simple operation latency: simple operations are integer adds, loads, stores, etc.

Complex operations are divides, cache misses.

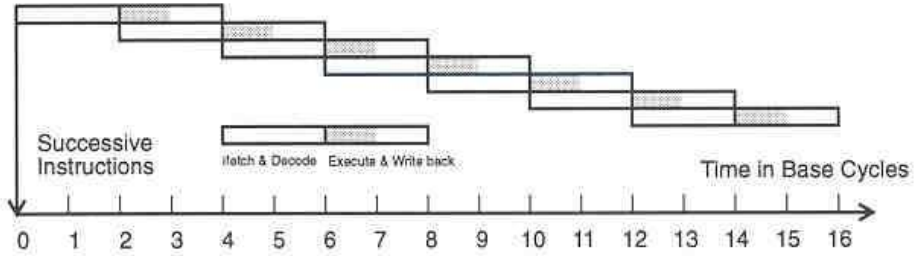
Resource Conflicts: two or more instructions demand use of the same functional unit at the same time.



(a) Execution in a base scalar processor



(b) Underpipelined with two cycles per instruction issue



(c) Underpipelined with twice the base cycle

Figure 4.2 Pipelined execution of successive instructions in a base scalar processor and in two underpipelined cases. Courtesy of Jouppi and Wall; reprinted from *Proc. ASPLOS*, ACM Press, 1989)

- A *base scalar processor*:
  - issues one instruction per cycle
  - has a one-cycle latency for a simple operation
  - has a one-cycle latency between instruction issues
  - can be fully utilized if instructions can enter the pipeline at a rate on one per cycle
- For a variety of reasons, instructions might not be able to be pipelines as agressively as in a base scalar processor. In these cases, we say the pipeline is *underpipelined*.
- CPI rating is 1 for an ideal pipeline. Underpipelined systems will have higher CPI ratings, lower clock rates, or both.

- The control unit generates control signals required for the *fetch, decode*, *ALU operation*, *memory access*, and *write result* phases of instruction execution

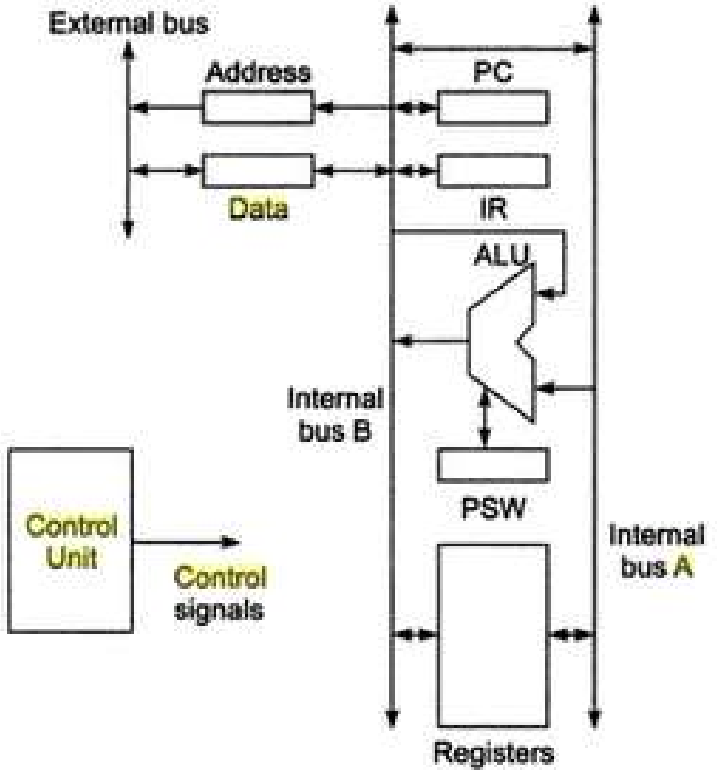


Fig. 4.3 Data path architecture and control unit of a scalar processor

4.1.2 INSTRUCTION-SET ARCHITECTURES

- The instruction set of a computer specifies the primitive commands or machine instructions that a programmer can use in programming the machine.
- The complexity of an instruction set is attributed to the instruction formats, addressing modes, general purpose registers, opcode specification and flow control mechanisms used.
- Two classes
  - RISC
  - CISC

COMPLEX INSTRUCTION SETS

- Simple instruction set – high cost of hardware
  - more and more functions were built into the hardware, making the instruction set large and complex.
- A typical CISC instruction set contains approximately 120 to 350 instructions using variable instruction/data formats,
- uses a small set of 8 to 24 *general-purpose registers* (GPRs),
- and executes a large number of memory reference operations based on more than a dozen addressing modes.

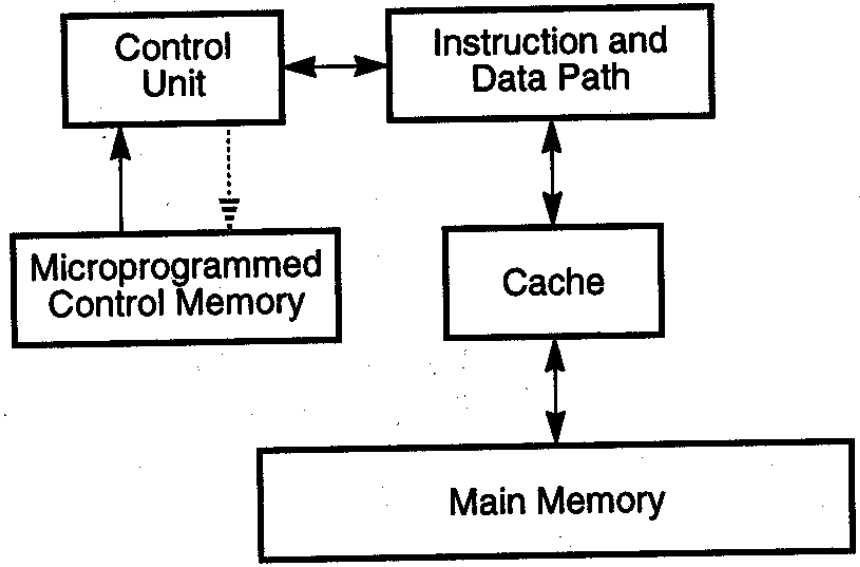
REDUCED INSTRUCTION SETS

- only 25% of the instructions of a complex instruction set are frequently used about 95% of the time. This implies that about 75% of hardware supported instructions often are not used at all.
  - Pushing rarely used instructions into software would vacate chip areas for building more powerful RISC

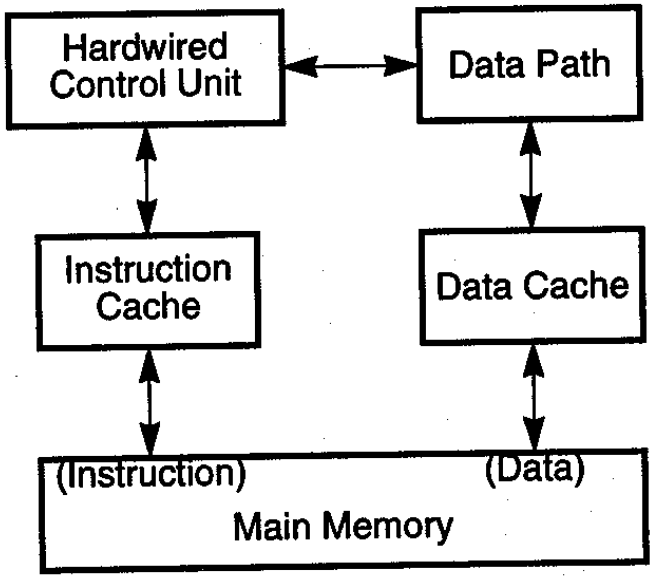
A RISC instruction set typically contains less than 100 instructions with a fixed instruction format (32 bits). Only three to five simple addressing modes are used. Most instructions are register-based. Memory access is done by load/store instructions only . A large register file (at least 32) is used to improve fast context switching among multiple users, and most instructions execute in one cycle with hardwired control.

ARCHITECTURAL DISTINCTIONS

- CISC
  - Unified cache for instructions and data (in most cases)
  - Microprogrammed control units and ROM in earlier processors (hard-wired controls units now in some CISC systems)
- RISC
  - Separate instruction and data caches
  - Hard-wired control units



(a) The CISC architecture with microprogrammed control and unified cache



(b) The RISC architecture with hardwired control and split instruction cache and data cache.

Table 4.1 *Characteristics of Typical CISC and RISC Architectures*

Architectural Characteristic	*Complex Instruction Set Computer (CISC)*	*Reduced Instruction Set Computer (RISC)*
Instruction-set size and instruction formats	Large set of instructions with variable formats (16–64 bits per instruction).	Small set of instructions with fixed (32-bit) format and most register-based instructions.
Addressing modes	12–24.	Limited to 3–5.
General-purpose registers and cache design	8–24 GPRs, originally with a unified cache for instructions and data, recent designs also use split caches.	Large numbers (32–192) of GPRs with mostly split data cache and instruction cache.
CPI	CPI between 2 and 15.	One cycle for almost all instructions and an average CPI < 1.5.
CPU Control	Earlier microcoded using control memory (ROM), but modern CISC also uses hardwired control.	Hardwired without control memory.

CISC SCALAR PROCESSORS

- A scalar processor executes with scalar data.
  - Early systems had only integer fixed point facilities.
- Modern machines have both fixed and floating point facilities, sometimes as parallel functional units.
- Many CISC scalar machines are underpipelined.
  - Underpipelined systems will have higher CPI ratings, lower clock rates, or both.
- Representative systems:
  - VAX 8600
  - Motorola MC68040
  - Intel Pentium

VAX 8600

- **VAX** : **Virtual Address Extension** (32-bit extension of the older 16-bit early versions)
- **Manufacturer**: **Digital Equipment Corporation (DEC)**.
- **First Model: VAX-11/780**
- VAX is a family of popular and influential computers implementing VAX Instruction Set Architecture.
- The **VAX 8600**, ("*Venus*",Oct 1984), had **increased performance(4.2 times VAX-11/785), I/O capacity, and included macro-pipelining and ECL (emitter coupled logic).**
  - Cycle Time of VAX 8600 CPU: **80 ns(12.5 MHz)**

FEATURES OF VAX 8600 PROCESSOR

- Implements **CISC architecture** with micro programmed control.
- **300 instructions** with **20 different addressing modes**.
- **Two functional units** for concurrent execution of instruction of integer and floating point instructions.
- **16GPRs**
- **Pipeline** is built with **six stages.**
- The instruction unit prefetches and decodes and handles branching operations.
- A transistor look aside buffer (TLB) is used in memory for fast generation of the physical address from the virtual address

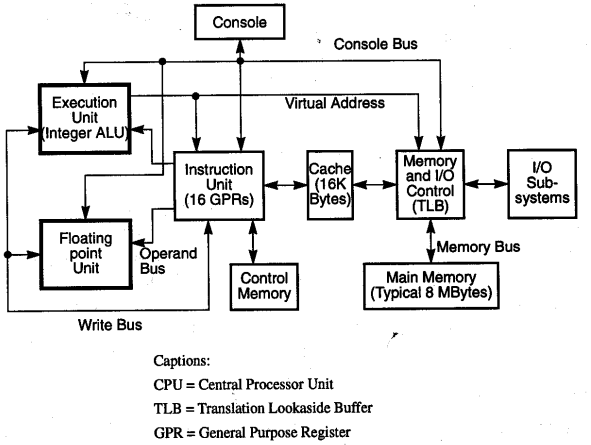


Figure 4.5 The VAX 8600 CPU, a typical CISC processor architecture. (Courtesy of Digital Equipment Corporation, 1985)

CISC Microprocessor Families In 1971, the Intel 4004 appeared as the first microprocessor based on a 4-bit ALU. Since then, Intel has produced the 8-bit 8008, 8080, and 8085. Intel's 16-bit processors appeared in 1978 as the 8086, 8088, 80186, and 80286. In 1985, the 80386 appeared as a 32-bit machine. The 80486 and Pentium are the latest 32-bit processors in the Intel 80x86 family.

Motorola produced its first 8-bit microprocessor, the MC6800, in 1974, then moved to the 16-bit 68000 in 1979, and then to the 32-bit 68020 in 1984. Then came the MC68030 and MC68040 in the Motorola MC680x0 family. National Semiconductor's 32-bit microprocessor NS32532 was introduced in 1988. These CISC microprocessor families have been widely used in the *personal computer* (PC) industry, with Intel x86 family dominating.

EX2: Motorola MC68040

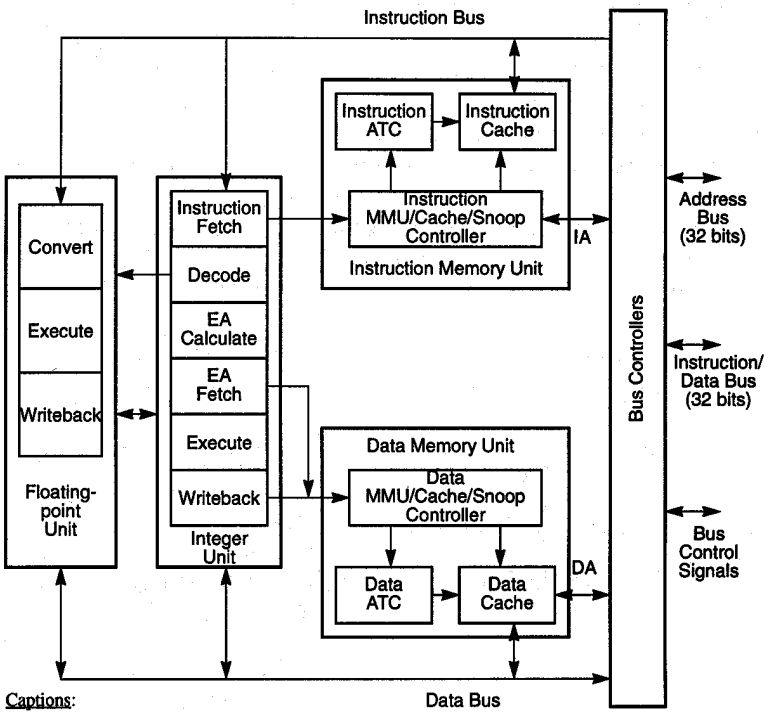


Figure 4.6 Architecture of the MC68040 processor. (Courtesy of Motorola Inc., 1991)

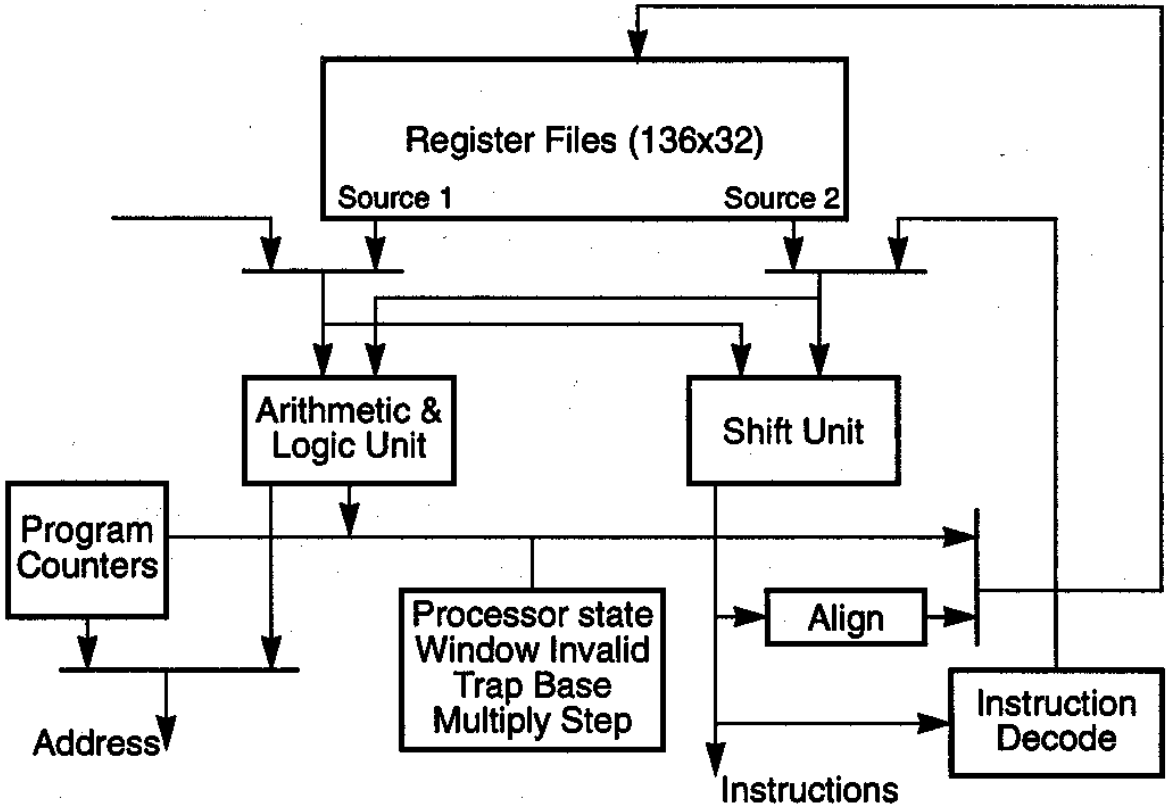
FEATURES :

- The MC68040 is a 0.8 µm HCMOS microprocessor containing more than 1.2 million transistor.
- It's implemented over 100 instructions using 16 general-purpose register.
- 4kb of data cache and 4kb of instruction cache with separate memory management unit (MMU's) supported by a address translation cache.
- Support 18 addressing modes, integer unit is organized in six stages of instruction pipeline, floating point consist of 3 stages.
- Separate instruction and data buses are provided both addressing and data buses are of 32bits width.
- The complete memory unit is provided with a virtual demand paged operating system.

RISC SCALAR PROCESSORS

- Designed to issue one instruction per cycle
- RISC and CISC scalar processors should have same performance if clock rate and program lengths are equal.
- RISC moves less frequent operations into software, thus dedicating hardware resources to the most frequently used operations.
- Representative systems:
  - Sun SPARC
  - Intel i860
  - Motorola M88100
  - AMD 29000

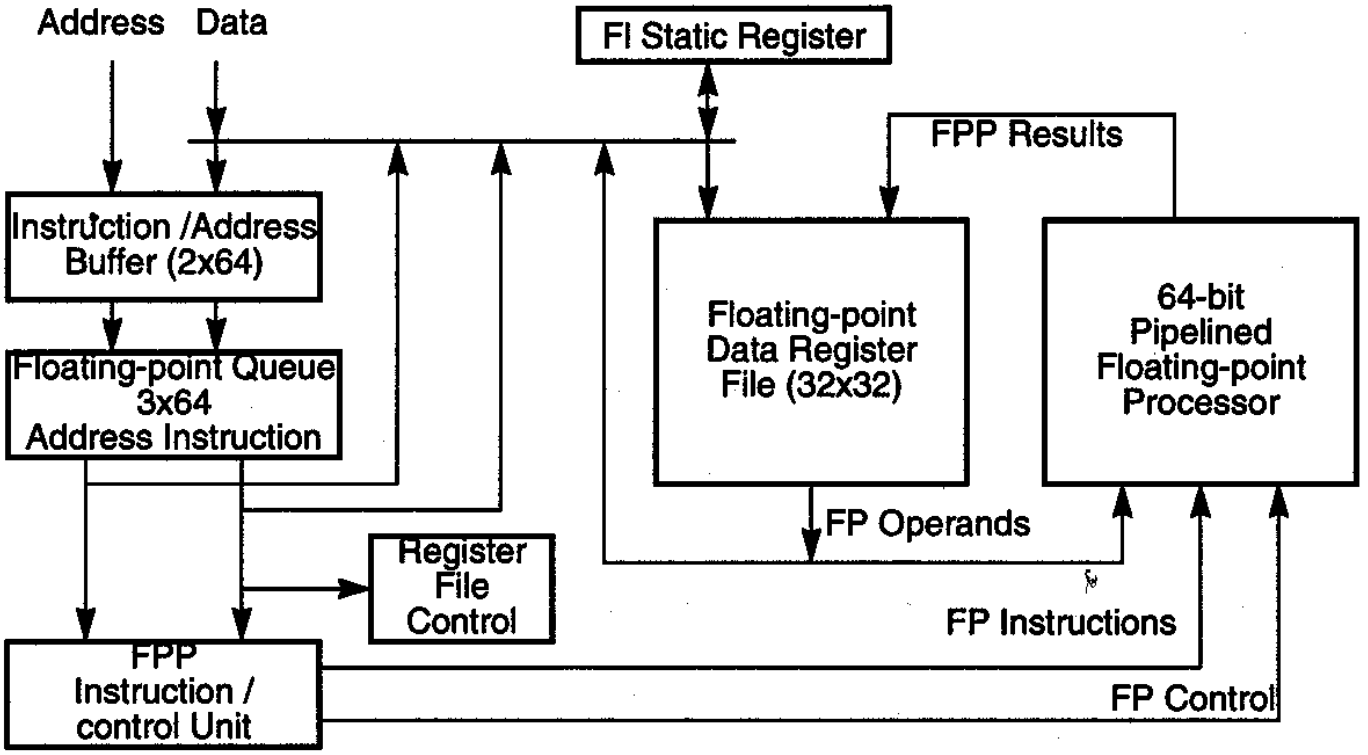
Ex1: The Sun Microsystems SPARC architecture



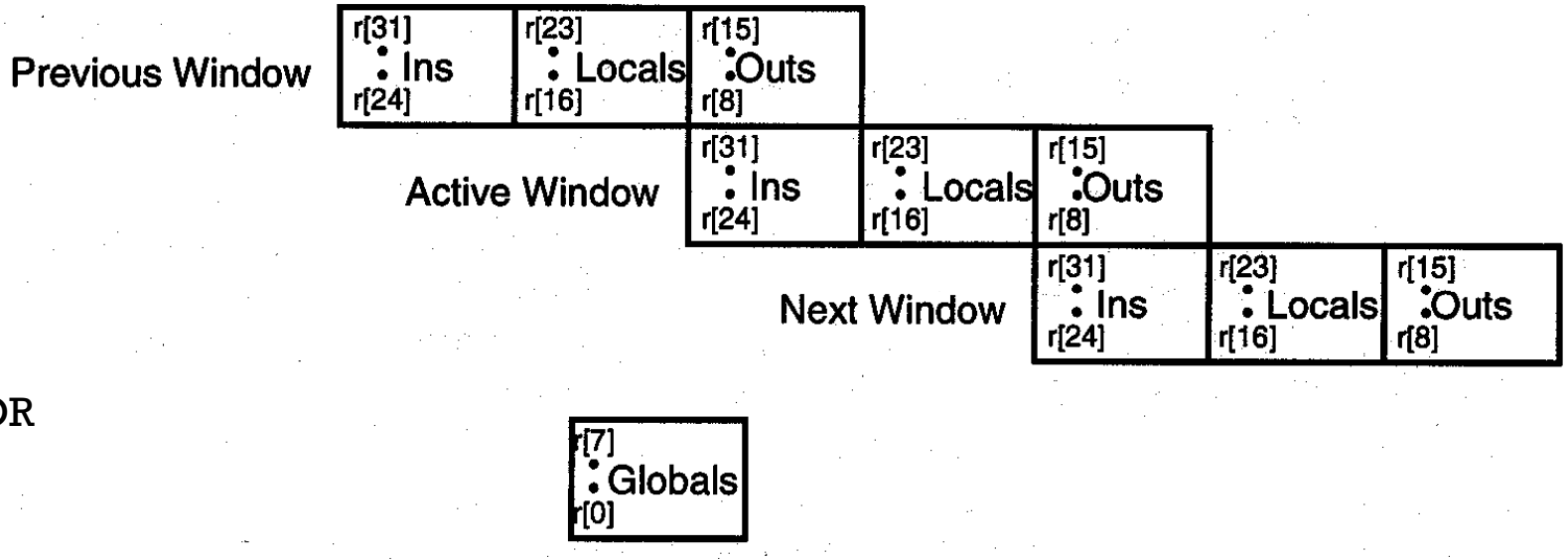
(a) The Cypress CY7C601 SPARC processor

- FPU ON SEPARATE CHIP
- 69 INSTR.

- 32 SINGLE PRECISION REG
- 16 DOUBLE PRECISION REG

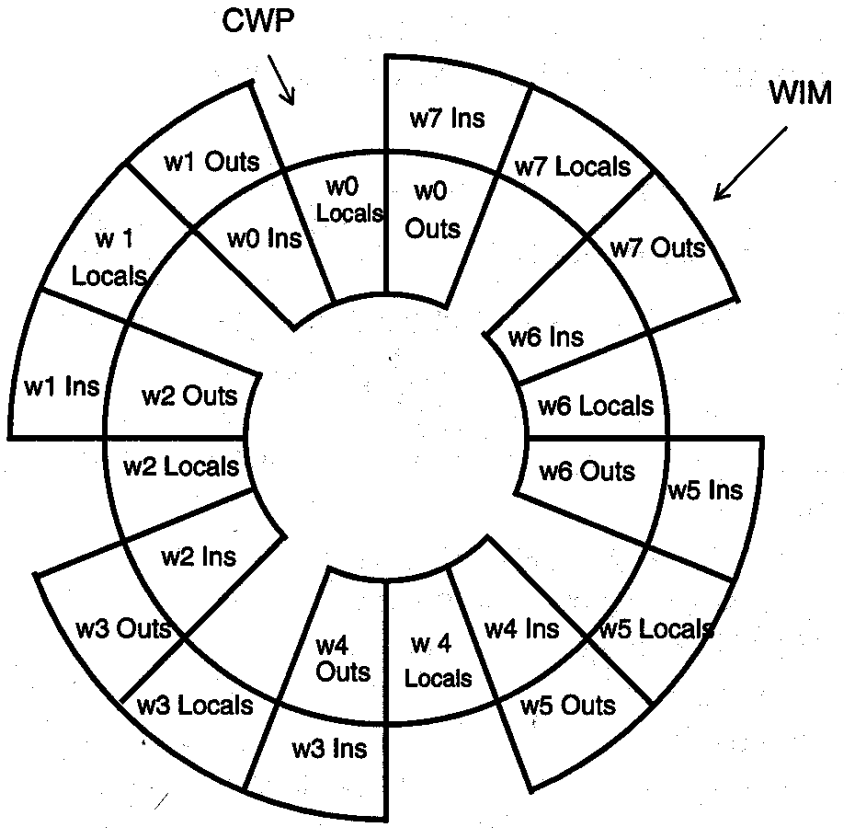


(b) The Cypress CY7C602 floating-point unit



(a) Three overlapping register windows and the global registers

- 24 WINDOW REG-ONLY FOR PROCEDURE
- 8 OVERLAPPING WINDOW
- 64 LOCAL REG
- 64 OVERLAPPED REG
- 8 GLOBAL REG
- TOTAL 136 REG
- REG WINDOW DIVIDED INTO 3 8 REG
- INS, LOCALS.OUTS
- ACTIVE WINDOW- CURRENTLY RUNNING PROC



(b) Eight register windows forming a circular stack

Figure 4.8 The concept of overlapping register windows in the SPARC architecture. (Courtesy of Sun Microsystems, Inc., 1987)

SPARCS AND REGISTER WINDOWS

- The SPARC architecture makes clever use of the logical procedure concept.
- Each procedure usually has some input parameters, some local variables, and some arguments it uses to call still other procedures.
- The SPARC registers are arranged so that the registers addressed as "Outs" in one procedure become available as "Ins" in a called procedure, thus obviating the need to copy data between registers.
- This is similar to the concept of a "stack frame" in a higher-level language.

EX2: INTEL I860 PROCESSOR ARCHITECTURE

- Introduced by Intel Corporation in 1989.
- It is a 64 bit RISC processor fabricated on a single chip containing more than one million transistors.
- There are 9 functional units interconnected by multiple data paths with widths ranging from 32 to 128 bits.
- All external or internal address buses are 32 bit wide.
- All external or internal data bus is 64 bits wide.

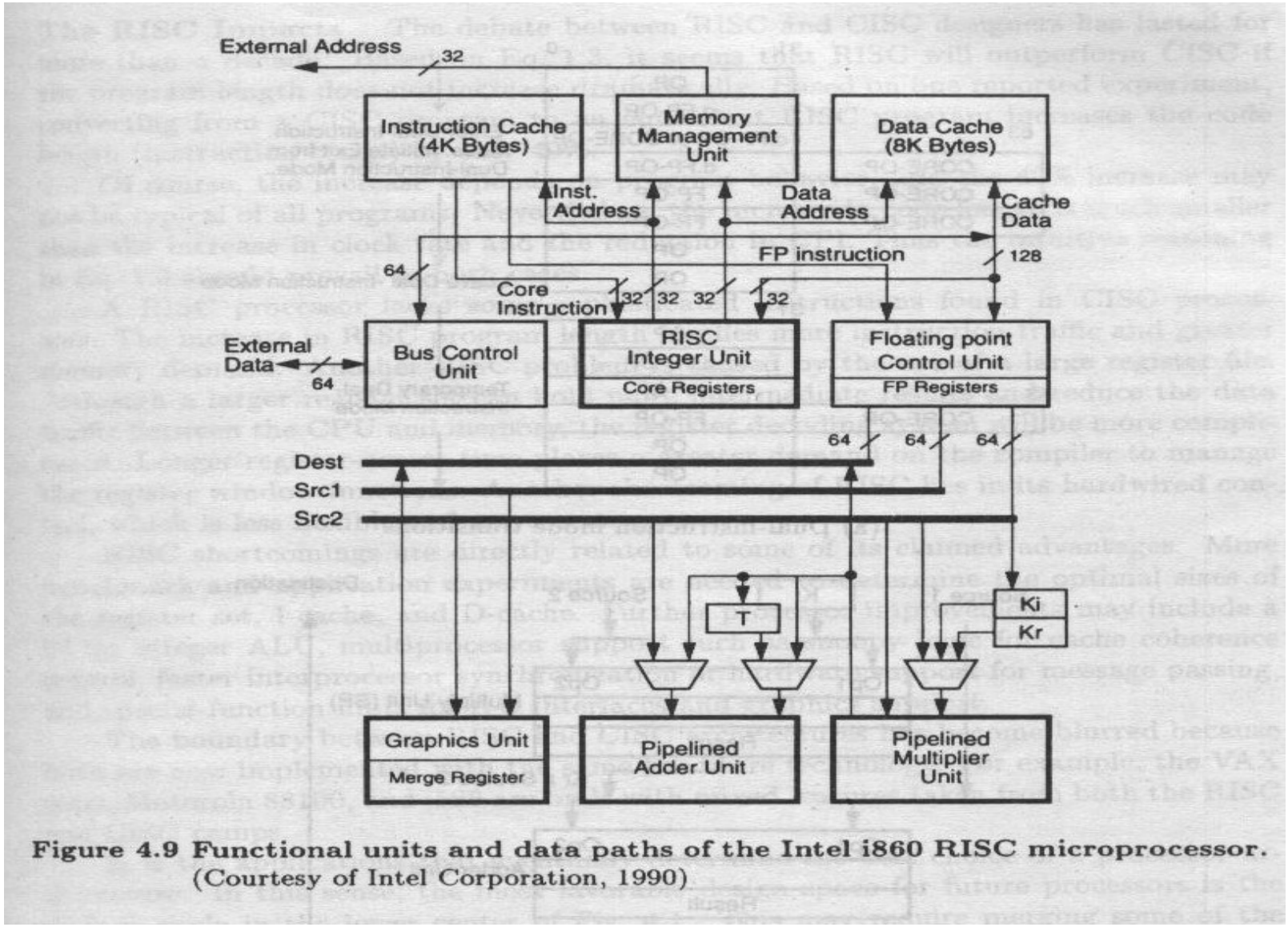


Figure 4.9 Functional units and data paths of the Intel i860 RISC microprocessor.
(Courtesy of Intel Corporation, 1990)

- Instruction cache has 4Kbytes organized as a two way set-associative memory with 32 bytes per cache block. It transfers 64 bits per clock cycle.

- Data cache is a two way set-associative memory of 8Kbytes.It transfers 128 bits per clock cycle. Write-back policy is used.

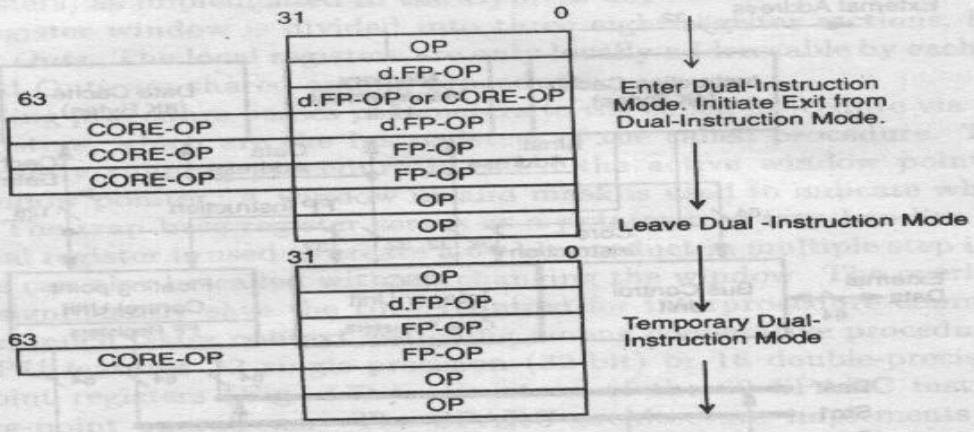
- Bus control co-ordinates the 64 bit data transfer between chip and outside world.

- MMU implements protected 4Kbyte paged virtual memory of 2^{32} bytes via TLB.

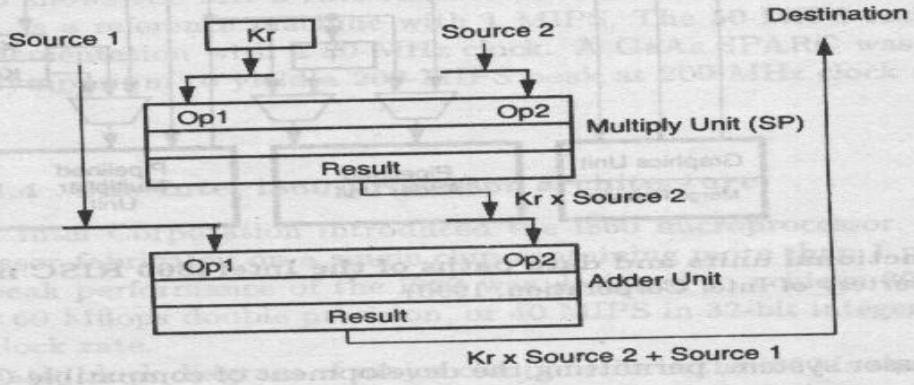
- There are two floating point units: multiplier-unit and adder-unit which can be used separately or simultaneously under coordination of the floating point control unit.

- Both integer unit and floating point control unit can execute concurrently.

- Graphics unit executes integer operations corresponding to 8,126,32 bit pixel data types.
- This unit supports three-dimensional drawing in a graphics frame buffer with color intensity, shading and hidden surface elimination.
- Merge register is used only by vector integer instructions.
- i860 executes 82 instructions including 42 RISC integer, 24 floating point,10 graphics and 6 assembler pseudo operations.
- All these instructions execute in one cycle.



(a) Dual-instruction mode transitions



(b) Dual operations in floating-point units

Figure 4.10 Dual instructions and dual operations in the i860 processor.

intensity shading, and hidden surface elimination. The merge

CISC VS. RISC

- CISC Advantages
  - Smaller program size (fewer instructions)
  - Simpler control unit design
  - Simpler compiler design
- RISC Advantages
  - Has potential to be faster
  - Many more registers
- RISC Problems
  - More complicated register decoding system
  - Hardwired control is less flexible than microcode

4.2 SUPERSCALAR AND VECTOR PROCESSORS

- A CISC or a RISC scalar processor can be improved with a *superscalar* or *vector* architecture.
- Scalar processors are those executing one instruction per cycle.
  - Only one instruction is issued per cycle, and
  - only one completion of instruction is expected from the pipeline per cycle.
- In a superscalar processor, multiple instructions are issued per cycle and multiple results are generated per cycle.
- A vector processor executes vector instructions on arrays of data;
  - each vector instruction involves a string of repeated operations, which are ideal for pipelining with one result per cycle.

SUPERSCALAR PROCESSORS

- These are designed to exploit more instruction-level parallelism in user programs.
- Only independent instructions can be executed in parallel without causing a wait state.
- The instruction-issue degree 2 to 5 in practice.

- ***Pipelining in Superscalar Processors*** The fundamental structure of a three-issue superscalar pipeline is illustrated in Fig. 4.11.
- Superscalar processors were originally developed as an alternative to vector processors,
  - with a view to exploit higher degree of instruction level parallelism.

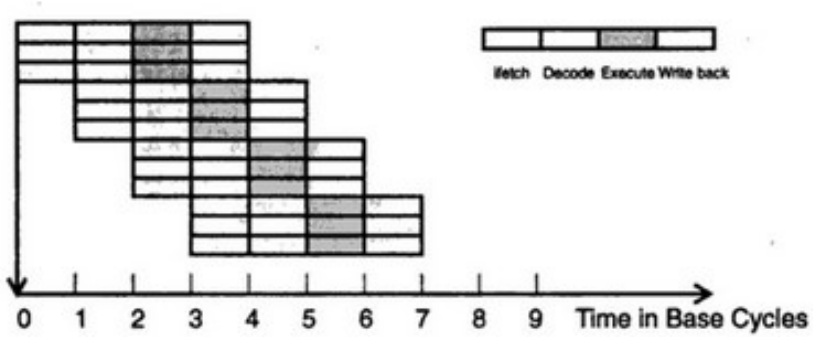


Figure 4.11 A superscalar processor of degree m = 3.

- A superscalar processor of degree *m* can issue *m* instructions per cycle.
- the base scalar processor, implemented either in RISC or CISC, has $m = 1$.
- In order to fully utilize a superscalar processor of degree *m*, *m* instructions must be executable in parallel.
- This situation may not be true in all clock cycles. In that case, some of the pipelines may be stalling in a wait state.

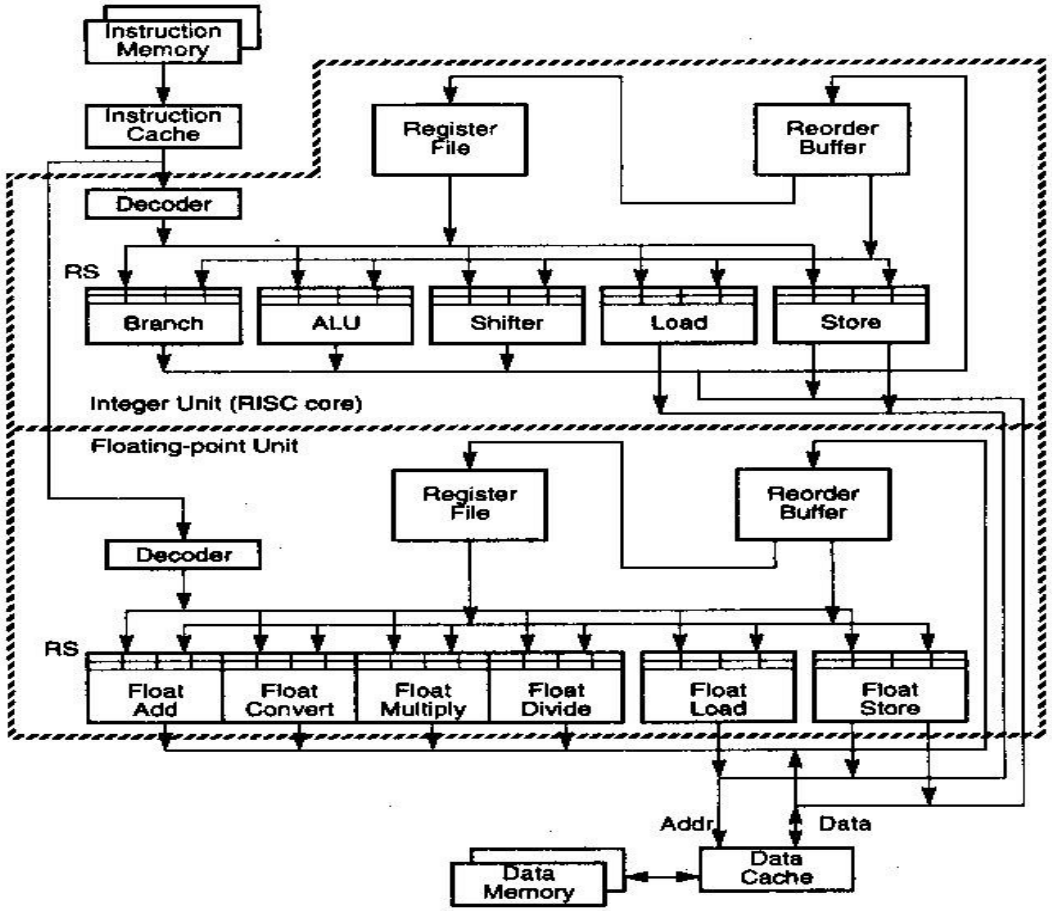


Figure 4.12 A typical superscalar RISC processor architecture consisting of an integer unit and a floating-point unit. (Courtesy of M. Johnson, 1991; reprinted with permission from Prentice-Hall, Inc.)

- A typical superscalar architecture for a RISC processor is shown in Fig. 4.12.
- Multiple instructions pipelines are used.
- Instruction cache supplies multiple instructions per fetch.
- Multiple functional units are built into the integer unit and into the floating –point unit.
- Multiple data buses exist among the functional units.
- IBM RS/6000,DEC 21064,Intel i960CA are examples of superscalar processors

- Due to the reduced CPI and higher clock rates used, most superscalar processors outperform scalar processors.

- The maximum number of instructions issued per cycle ranges from two to five in these superscalar processors.

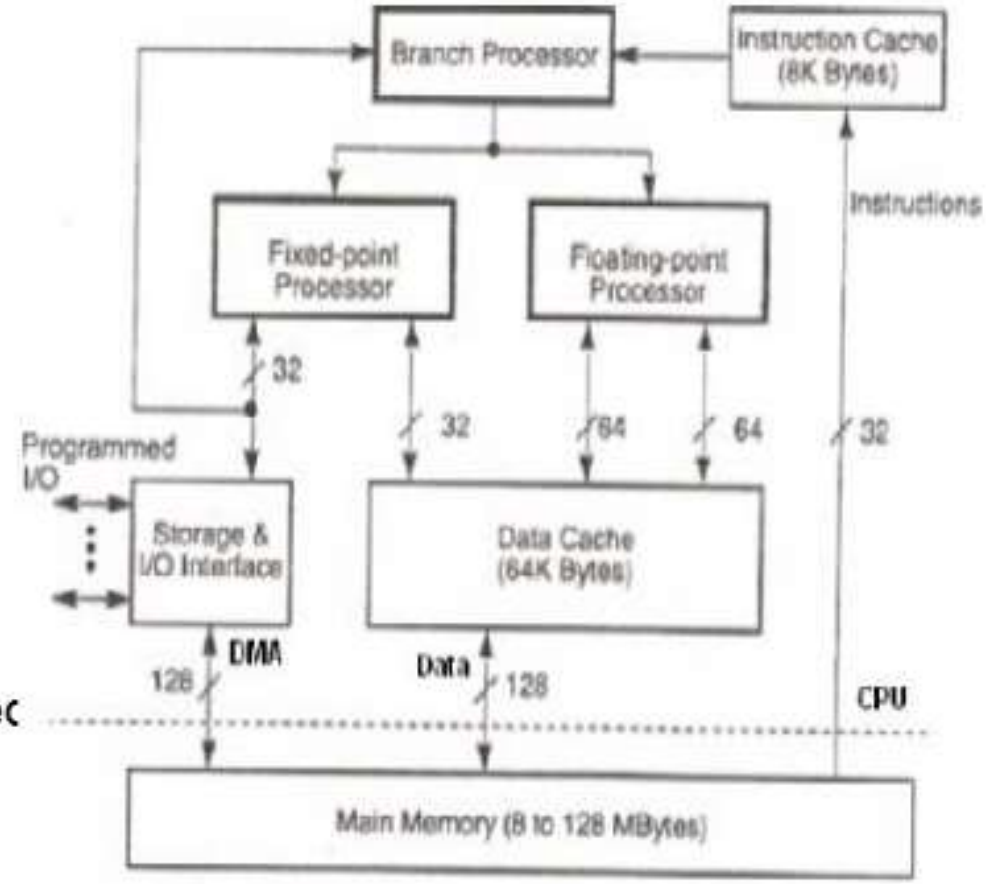
- Register files in IU and FPU each have 32 registers. Both IU and FPU are implemented on the same chip.

- Superscalar degree is low due to limited instruction parallelism that can be exploited in ordinary programs.

- Reservation stations and reorder buffers can be used to establish instruction windows.
- The purpose is to support instruction look ahead and internal data forwarding, which is needed to schedule multiple instructions through the multiple pipelines simultaneously.

Example: IBM RS/6000

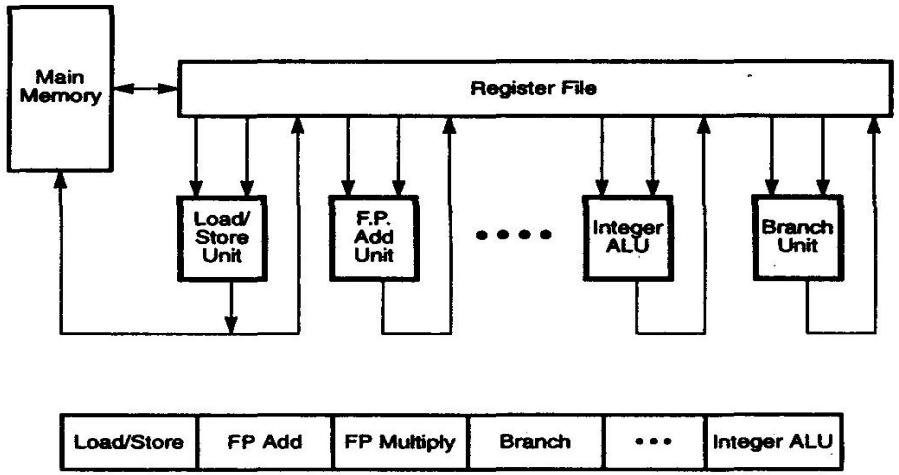
- IBM announced this superscalar RISC system in 1990.
- There are 3 parallel functional units; branch processor, fixed-point unit, and floating-point unit.
- The branch processor can arrange the execution of up to 5 IPC.
- It is hardwired rather than micro-programmed control unit.
- The system uses a number of wide buses. These will provide the high instruction and data bandwidths required for superscalar implementation.
- This system design is optimized to perform well in numerically intensive scientific and engineering applications.



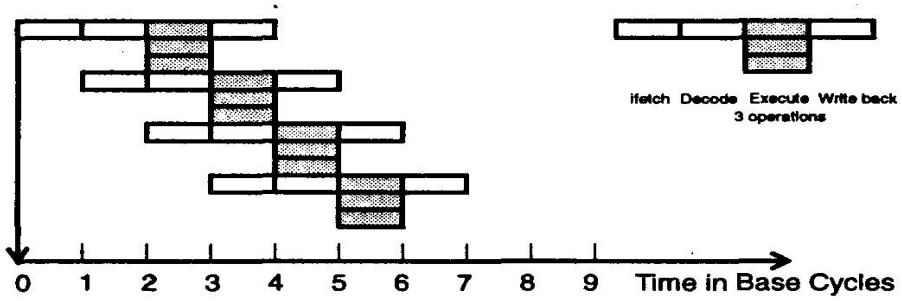
The POWER architecture of the IBM RISC System/6000 superscalar processor.

4.2.2 VLIW ARCHITECTURE

- This architecture is generalized from two well established concepts:
  - horizontal micro coding and superscalar processing.
- The instruction word has hundreds of bits in length.
- Multiple functional units are used concurrently .
- All functional units share the use of common large register file.



(a) A typical VLIW processor and instruction format



(b) VLIW execution with degree $m = 3$

Figure 4.14 The architecture of a very long instruction word (VLIW) processor and its pipeline operations. (Courtesy of Multiflow Computer, Inc., 1987)

- Different fields of the long instruction word carry the opcodes to be dispatched to different functional units.

- Programs written in conventional short instruction words(32 bits) must be compacted together to form the VLIW instructions.

PIPELINING IN VLIW PROCESSORS

- VLIW machines behave much like superscalar machines with three differences.

1. Decoding of VLIW instructions is easier than that of superscalar instructions.
2. Code density of superscalar machine is better when the available instruction-level parallelism is less than that exploitable by the VLIW machine.
    - This is because the fixed VLIW format includes bits for non executable operations, while the superscalar processor issues only executable instructions.

3. Superscalar machine can be object-code-compatible with a large family of nonparallel machines, but VLIW machine exploiting different amounts of parallelism would require different instruction sets.

- Instruction parallelism and data movement in a VLIW architecture are specified at compile time.
- Run-time scheduling and synchronization are thus completely eliminated.

One can view VLIW processor as an extreme of superscalar processor in which all independent or unrelated operations are already synchronously compacted together in advance.

VLIW OPPORTUNITIES:

- Random parallelism among scalar operations is exploited instead of regular or synchronous parallelism as in a vectorized supercomputer or in an SIMD computer.
- Success depends heavily on the efficiency in code compaction.
- VLIW architecture is totally incompatible with that of any conventional general-purpose processor.

- Instruction parallelism embedded in the compacted code may require a different latency to be executed by different functional units even though the instructions are issued at the same time.
- Therefore, different implementations of the same VLIW architecture may not be binary-compatible with each other.
- By explicitly encoding parallelism in the long instruction, a VLIW processor can eliminate the hardware or software needed to detect parallelism.

- Advantage of VLIW is simplicity in hardware structure and instruction set.

- It can perform well in scientific applications where the program behavior is more predictable.

- In general-purpose applications, the architecture may not be able to perform well. Due to the lack of compatibility with conventional hardware and software, the architecture is not entered the mainstream of computers.

- The dependence on trace-scheduling and code-compaction has prevented it from gaining acceptance in the commercial world.

VECTOR PROCESSORS

- A vector processor is a coprocessor designed to perform vector computations.
  - a processor that is able to process sequences of data with a single instruction.
- A vector is a one-dimensional array of data items (each of the same data type).
- Vector processors are often used in multipipelined supercomputers.
- Architectural types include:
  - register-to-register (with shorter instructions and register files)
  - memory-to-memory (longer instructions with memory addresses)

REGISTER-TO-REGISTER VECTOR INSTRUCTIONS

- Assume V_i is a vector register of length n, s_i is a scalar register, M(1:n) is a memory array of length n, and "o" is a vector operation.
- Typical instructions include the following
  - $V_1 \, o \, V_2 \rightarrow V_3$ (element by element operation)
  - $s_1 \, o \, V_1 \rightarrow V_2$ (scaling of each element)
  - $V_1 \, o \, V_2 \rightarrow s_1$ (binary reduction - i.e. sum of products)
  - $M(1{:}n) \rightarrow V_1$ (load a vector register from memory)
  - $V_1 \rightarrow M(1{:}n)$ (store a vector register into memory)
  - $o \, V_1 \rightarrow V_2$ (unary vector -- i.e. negation)
  - $o \, V_1 \rightarrow s_1$ (unary reduction -- i.e. sum of vector)

MEMORY-TO-MEMORY VECTOR INSTRUCTIONS

- Tpyical memory-to-memory vector instructions (using the same notation as given in the previous slide) include these:
  - $M_1(1{:}n)$ o $M_2(1{:}n) \rightarrow M_3(1{:}n)$ (binary vector)
  - s_1 o $M_1(1{:}n) \rightarrow M_2(1{:}n)$ (scaling)
  - o $M_1(1{:}n) \rightarrow M_2(1{:}n)$ (unary vector)
  - $M_1(1{:}n)$ o $M_2(1{:}n) \rightarrow M(k)$ (binary reduction)

PIPELINES IN VECTOR PROCESSORS

- Vector processors can usually effectively use large pipelines in parallel, the number of such parallel pipelines effectively limited by the number of functional units.
- As usual, the effectiveness of a pipelined system depends on the availability and use of an effective compiler to generate code that makes good use of the pipeline facilities.

SYMBOLIC PROCESSORS

Symbolic Processing has been used in many areas like theorem proving, pattern recognition, expert systems and so on. Herein, we have a modern and non-algorithmic approach of problem solving. Symbolic processors include prolog processors Lisp processors or symbolic manipulators. It depends on various features like how knowledge is represented? What common operations are to be performed? What are the properties of algorithms? And so on. In symbolic processing, we deal with logic programs, lists, objects, scripts, net, frames and neural networks. No floating-point operations are done here.

For example : Symbolics 3600 Lisp processor

4.3 HIERARCHICAL MEMORY TECHNOLOGY

- Storage devices such as *registers*, *caches*, *main memory*, *disk devices*, and *backup storage* are often organized as a hierarchy as depicted in Fig. 4.17.

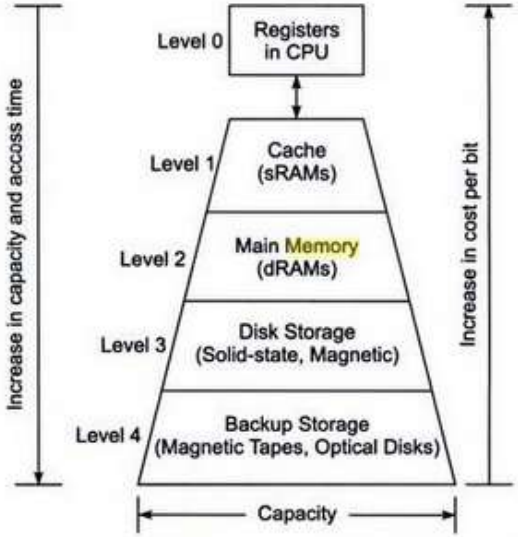


Fig. 4.17 A four-level memory hierarchy with increasing capacity and decreasing speed and cost from low to high levels

- Each level is characterized by five parameters:
  - access time t_i (round-trip time from CPU to ith level)
  - memory size s_i (number of bytes or words in the level)
  - cost per byte c_i (estimated by the product c_i s_i)
  - transfer bandwidth b_i (rate of transfer between levels)
  - unit of transfer x_i (grain size for transfers)

MEMORY GENERALITIES

- It is almost always the case that memories at lower-numbered levels, when compare to those at higher-numbered levels
  - are faster to access,
  - are smaller in capacity,
  - are more expensive per byte,
  - have a higher bandwidth, and
  - have a smaller unit of transfer.
- In general, then, $t_{i-1} < t_i$, $s_{i-1} < s_i$, $c_{i-1} > c_i$, $b_{i-1} > b_i$, and $x_{i-1} < x_i$.

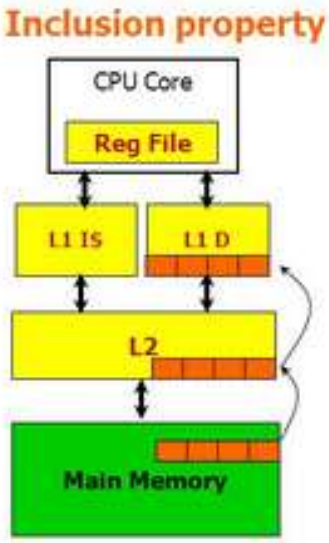
MEMORY CHARACTERISTIC OF A TYPICAL MAINFRAME COMPUTER IN 1993

Memory Characteristics	CPU Register	Cache	Main Memory	Disk Storage	Tape Unit
Access time t_i	10 ns	25 ~ 40 ns	60 ~ 100 ns	12 ~ 20 ms	2 ~ 20 min
Capacity s_i	512 bytes	128 Kbytes	512 Mbytes	60 ~ 228 GB	0.5 ~ 2 TB
Bandwidth b_i	400~800 MB/s	250~400 MB/s	80~133 MB/s	3~5 MB/s	0.18 ~ 0.23 MB/s
Unit of transfer x_i	4 ~ 7 bytes per word	32 bytes per block	0.5 ~ 1 Kbytes per page	5 ~ 512 Kbytes per file	Backup storage
Allocation management	Compiler assignment	Hardware Control	Operating system	Operating system / user	Operating system / user

Information stored in a memory hierarchy (M1,M2,…Mn) satisfies 3 important properties:

- Inclusion
- Coherence
- locaity

THE INCLUSION PROPERTY



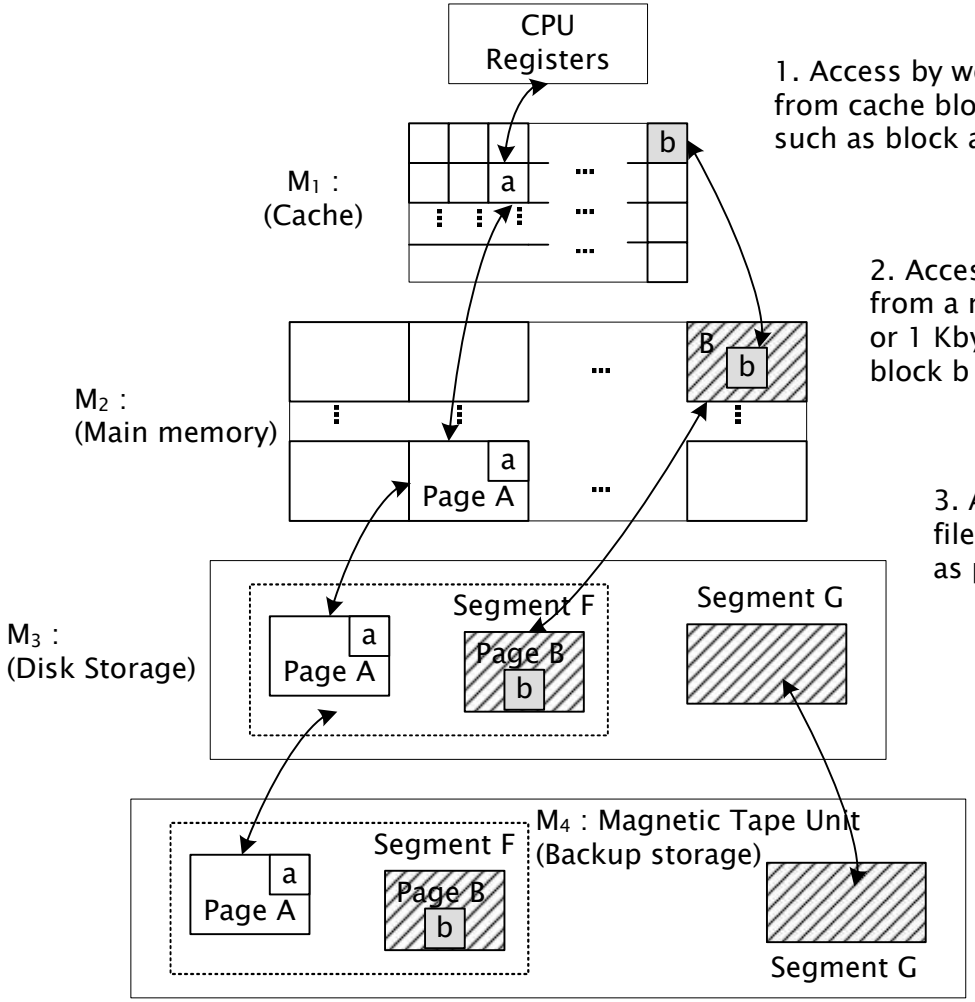
- The inclusion property is stated as:

  $M_1 \subset M_2 \subset \ldots \subset M_n$

- The set inclusion relationship implies that all information items are originally stored in the outermost level *Mn*. During the processing, subsets of *Mn* are copied into *Mn*–1. Similarly, subsets of *Mn*–1 are copied into *Mn*–2, and so on.

- The inverse, however, is not necessarily true. That is, the presence of a data item in level M_{i+1} does not imply its presence in level M_i. We call a reference to a missing item a "miss."

Inclusion property and data transfer



1. Access by word (4 Bytes) from cache block of 32 bytes, such as block a

2. Access by block (32 Bytes) from a memory page of 32 block or 1 Kbytes, such as block b from page B

3. Access by page (1 Kbytes) from a file consisting of many pages, such as page A and page B in segment F

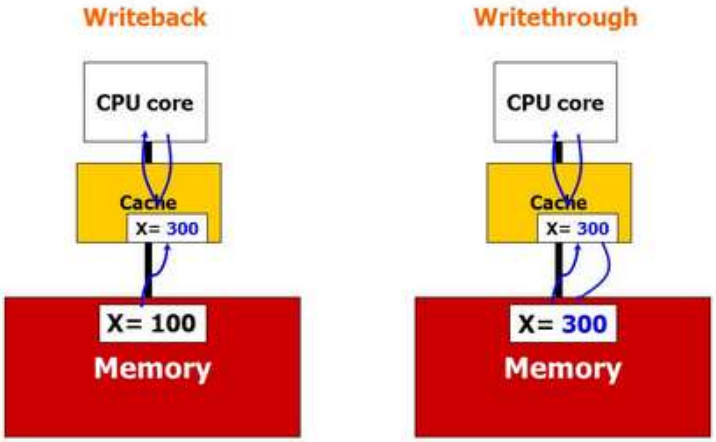
4. Segment transfer with different number of pages

THE COHERENCE PROPERTY

Coherence Property The *coherence property* requires that copies of the same information item at successive memory levels be consistent. If a word is modified in the cache, copies of that word must be updated immediately or eventually at all higher levels. The hierarchy should be maintained as such. Frequently used information is often found in the lower levels in order to minimize the effective access time of the memory hierarchy. In general, there are two strategies for maintaining the coherence in a memory hierarchy.

COHERENCE STRATEGIES

- Write-through
  - As soon as a data item in M_i is modified, immediate update of the corresponding data item(s) in M_{i+1}, M_{i+2}, … M_n is required. This is the most aggressive (and expensive) strategy.
- Write-back
  - The update of the data item in M_{i+1} corresponding to a modified item in M_i is not updated until it (or the block/page/etc. in M_i that contains it) is replaced or removed. This is the most efficient approach, but cannot be used (without modification) when multiple processors share M_{i+1}, …, M_n.



LOCALITY OF REFERENCES

- In most programs, memory references are assumed to occur in patterns that are strongly related (statistically) to each of the following:
  - *Temporal locality* – if location M is referenced at time t, then it (location M) will be referenced again at some time t+Δt.
  - *Spatial locality* – if location M is referenced at time t, then another location M$\pm\Delta$m will be referenced at time t+Δt.
  - *Sequential locality* – if location M is referenced at time t, then locations M+1, M+2, … will be referenced at time t+Δt, t+Δt', etc.
- In each of these patterns, both Δm and Δt are "small."
- Hennessy&Patterson suggest that 90 percent of the execution time in most programs is spent executing only 10 percent of the code.

Temporal Locality

Repeatedly referring to same data in a short time span

Spatial Locality

Referring to data that is close together in memory

Sequential Locality

Referring to data that is arranged linearly in memory

Locality Example

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

Data references

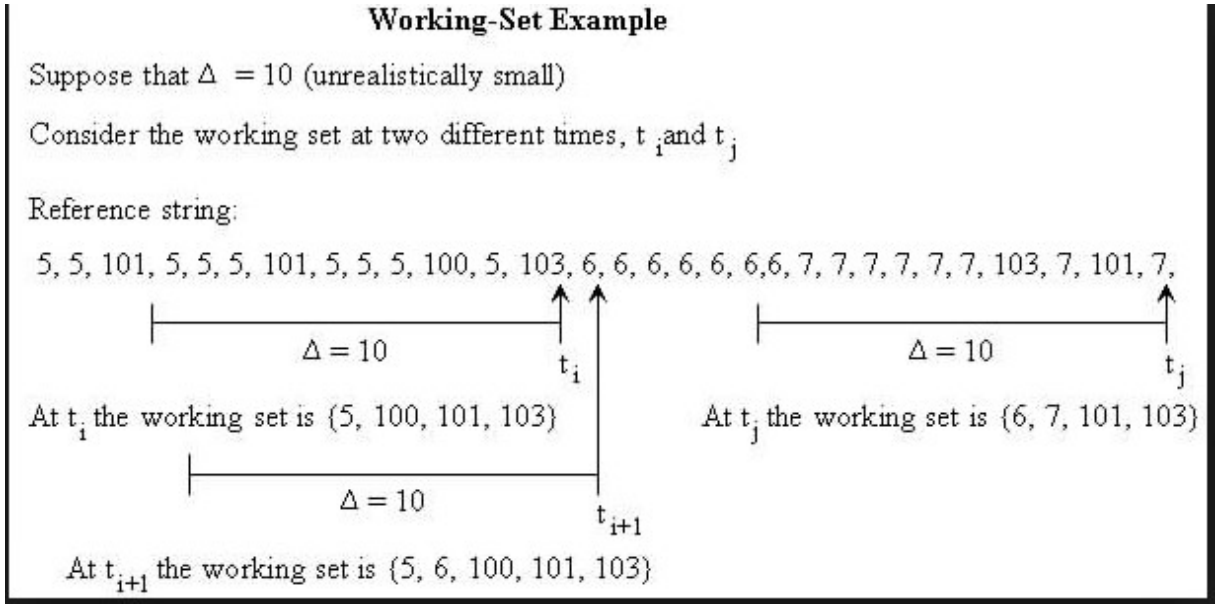
- Reference array elements in succession (stride-1 reference pattern). — Spatial locality
- Reference variable `sum` each iteration. — Temporal locality

Instruction references

- Reference instructions in sequence. — Spatial locality
- Cycle through loop repeatedly. — Temporal locality

WORKING SETS

- The set of addresses (bytes, pages, etc.) referenced by a program during the interval from t to t+ω, where ω is called the *working set parameter*, changes slowly.
- This set of addresses, called the *working set*, should be present in the higher levels of M if a program is to execute efficiently (that is, without requiring numerous movements of data items from lower levels of M). This is called the *working set principle*.



HIT RATIOS

- When a needed item (instruction or data) is found in the level of the memory hierarchy being examined, it is called a *hit*. Otherwise (when it is not found), it is called a *miss* (and the item must be obtained from a lower level in the hierarchy).
- The *hit ratio*, *h*, for M_i is the probability (between 0 and 1) that a needed data item is found in level memory M_i.
- The *miss ratio* is obviously just $1-h_i$.
- We assume $h_0 = 0$ and $h_n = 1$.
  - CPU access always M1 first and access to the outermost memory Mn is always hit.

ACCESS FREQUENCIES

- The access frequency f_i to level M_i is
  $fi = (1-h_1) \times (1-h_2) \times \ldots \times h_i$.

- Note that $f_1 = h_1$, and

$$\sum_{i=1}^{n} f_i = 1$$

EFFECTIVE ACCESS TIMES

- There are different penalties associated with misses at different levels in the memory hierarcy.
  - A cache miss is typically 2 to 4 times as expensive as a cache hit (assuming success at the next level).
  - A page fault (miss) is 3 to 4 magnitudes as costly as a page hit.
- The effective access time of a memory hierarchy can be expressed as

$$T_{eff} = \sum_{i=1}^{n} f_i \cdot t_i$$

$$= h_1 t_1 + (1-h_1) h_2 t_2 + \cdots + (1-h_1)(1-h_2)\cdots(1-h_{n-1}) h_n t_n$$

- The first few terms in this expression dominate, but the effective access time is still dependent on program behavior and memory design choices.

Hierarchy Optimization The total cost of a memory hierarchy is estimated as follows:

$$C_{\text{total}} = \sum_{i=1}^{n} c_i \cdot s_i \qquad (4.4)$$

Example 1. *Consider a three level memory hierarchy ($M_1 - M_3$) as shown below :*

Memory level	*Access time*	*Capacity*	*Cost/KB*
M_1 *(cache)*	$t_1 = 25$ *ns*	$S_1 = 512$ *KB*	$C_1 = \$1.25$
M_2 *(MM)*	$t_2 = ?$	$S_2 = 32$ *MB*	$C_2 = \$0.2$
M_3 *(Disk)*	$t_3 = 4$ *ms*	$S_3 = ?$	$C_3 = \$0.0002$

It is desired that :

$$T_{eff} = 10.04\ \mu s$$
$$h_1 = 0.98\ (for\ M_1)$$
$$h_2 = 0.9\ (for\ M_2).$$

But $C_{total} \leq \$15000$.
Find S_3, t_2 *(marked as ? in table) ?*

Solution. We know that the cost of memory hierarchy is given by the formula :

$$C = C_1S_1 + C_2S_2 + C_3S_3 \leq 15000$$

or $1.25\ (512) + 0.2\ (32) + 0.0002\ (S_3) = 15000$

or $S_3 =$ **39.8 Gbytes**

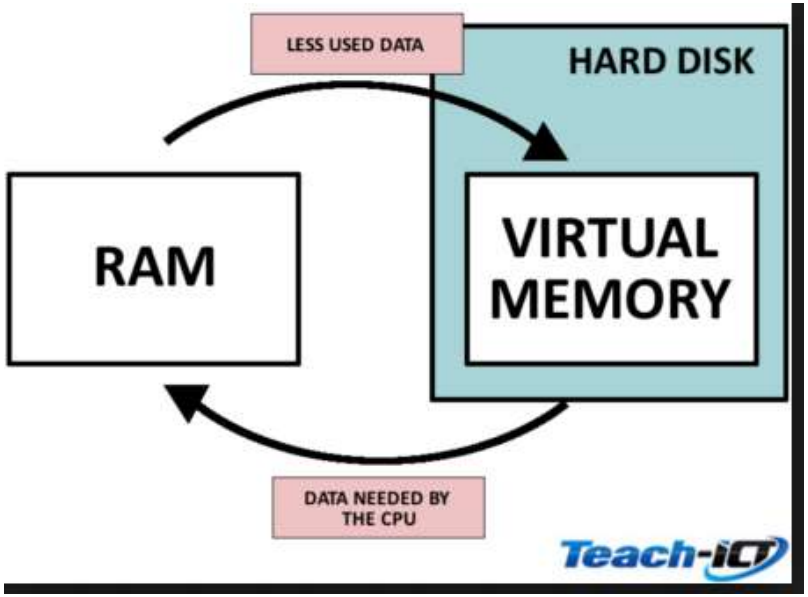
Now

$$T_{eff} = h_1t_1 + (1 - h_1)h_2t_2 + (1 - h_1)\ (1 - h_2)h_3t_3 \leq 10.04$$

or $10.04 \times 10^{-6} = 0.98 \times 25 \times 10^{-9} + 0.02 \times 0.9 \times t_2 + 0.02 \times 0.1 \times 1 \times 4 \times 10^{-3}$

or $t_2 =$ **903 ns.**

4.4 VIRTUAL MEMORY



VIRTUAL MEMORY MODELS

- The main memory is considered the *physical memory* in which multiple running programs may reside.
  - However, the limited-size physical memory cannot load in all programs fully and simultaneously.
- The *virtual memory* concept was introduced to alleviate this problem.
  - The idea is to expand the use of the physical memory among many programs with the help of an auxiliary (backup) memory such as disk arrays.
- Only active programs or portions of them become residents of the physical memory at one time.
  - Active portions of programs can be loaded in and out from disk to physical memory dynamically under the coordination of the operating system.
- To the users, virtual memory provides almost unbounded memory space to work with. Without virtual memory, it would have been impossible to develop the multiprogrammed cr time-sharing computer systems that are in use today.

Address Spaces Each word in the physical memory is identified by a unique *physical address*. All memory words in the main memory form a *physical address space*. *Virtual addresses* are those used by machine instructions making up an executable program.

Logical address:-

a)Logical address is address generated by CPU during execution

b) It is virtual address and user has access to use only logical address known as virtual address.

c) Set of all logical address is known as logical address space.

d) This address act as reference in creating physical address

e) It deals with compile time address binding

Physical address:-

a) Physical Address refers to location in memory unit(the one that is loaded into memory).

b) It cannot be viewed by user directly.

c) Memory management unit computes the physical address

d) Set of all physical address is physical address space.

e) It deals with load time address binding.

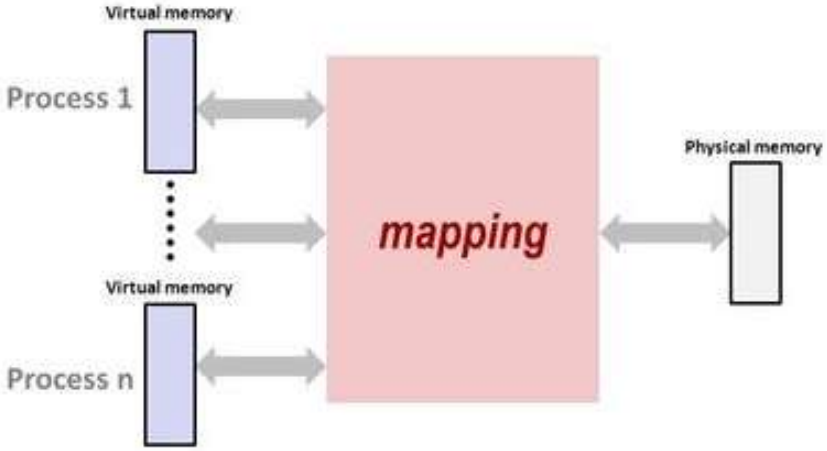
VIRTUAL TO PHYSICAL MAPPING

- The mapping from virtual to physical addresses can be formally defined as follows:

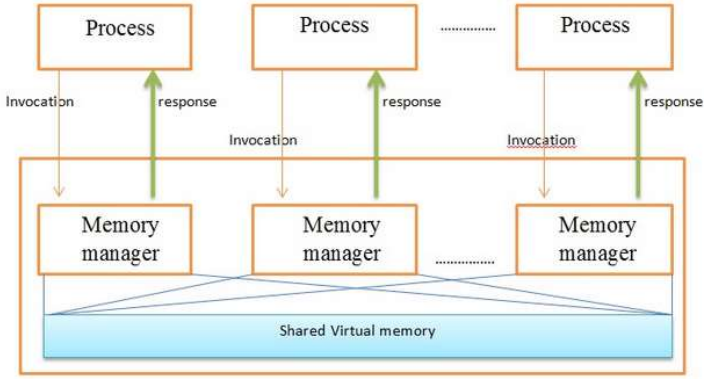
$$f_t v = \begin{cases} m, & \text{if } m \in M \text{ has been allocated to store} \\ & \text{the data identified by virtual address } m \\ \varnothing & \text{if data } v \textit{ is} \text{ missing in } M \end{cases}$$

- The mapping returns a physical address if a *memory hit* occurs. If there is a *memory miss*, the referenced item has not yet been brought into primary memory.

TWO VIRTUAL MEMORY MODELS



a)Private virtual memory



b)Shared virtual memory

PRIVATE VIRTUAL MEMORY

- In this scheme, each processor has a separate virtual address space, but all processors share the same physical address space.
- Example
  - VAX/11 and in most UNIX systems
- Advantages:
  - Small processor address space
  - Protection on a per-page or per-process basis
  - Private memory maps, which require no locking
- Disadvantages
  - The synonym problem – different virtual addresses in different/same virtual spaces point to the same physical page
  - The same virtual address in different virtual spaces may point to different pages in physical memory

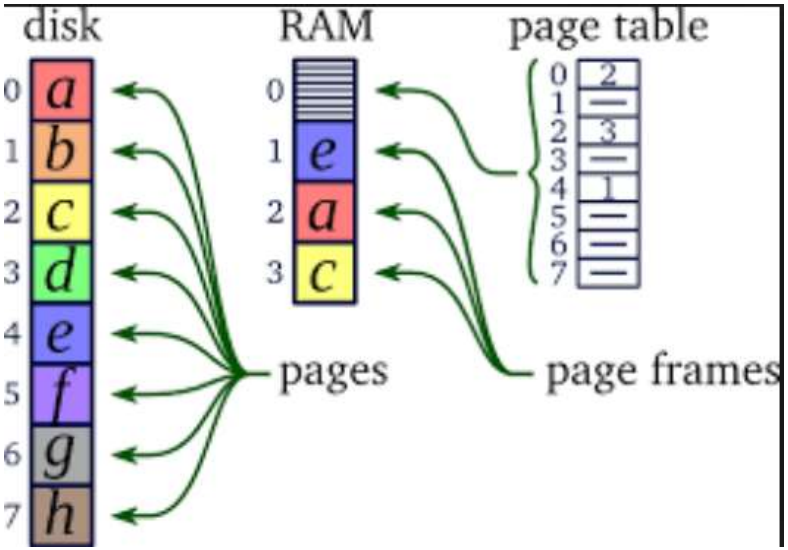
SHARED VIRTUAL MEMORY

- All processors share a single shared virtual address space, with each processor being given a portion of it.
- Some of the virtual addresses can be shared by multiple processors.
- Example
  - IBM801, RT, RP3, System 38, the HP Spectrum, the Stanford Dash, MIT Alewife, Tera, etc
- Advantages:
  - All addresses are unique
  - Synonyms are not allowed
- Disadvantages
  - Processors must be capable of generating large virtual addresses (usually > 32 bits)
  - Since the page table is shared, mutual exclusion must be used to guarantee atomic updates
  - Segmentation must be used to confine each process to its own address space
  - The address translation process is slower than with private (per processor) virtual memory

TLB, PAGING, AND SEGMENTATION

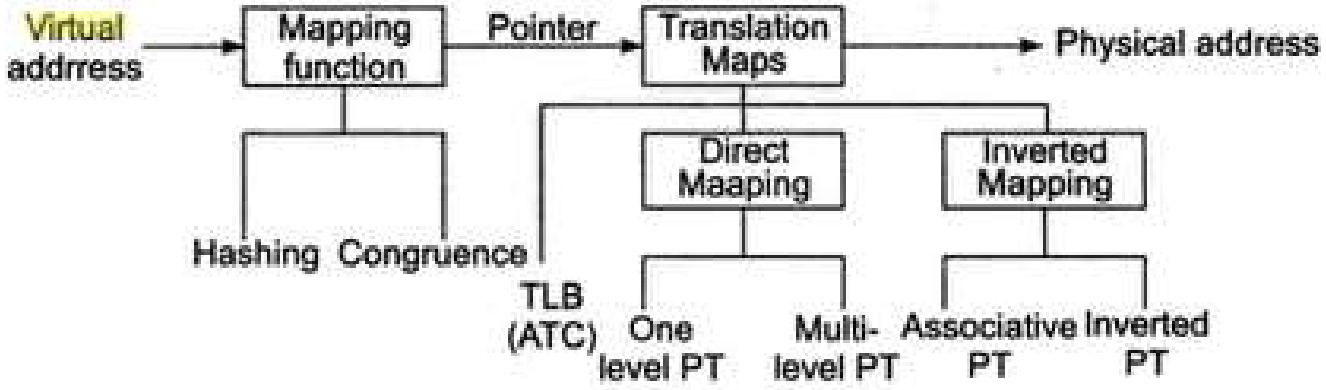
MEMORY ALLOCATION

- Both the virtual address space and the physical address space are divided into fixed-length pieces.
  - In the virtual address space these pieces are called *pages*.
  - In the physical address space they are called *page frames*.
- The purpose of memory allocation is to allocate pages of virtual memory using the page frames of physical memory.

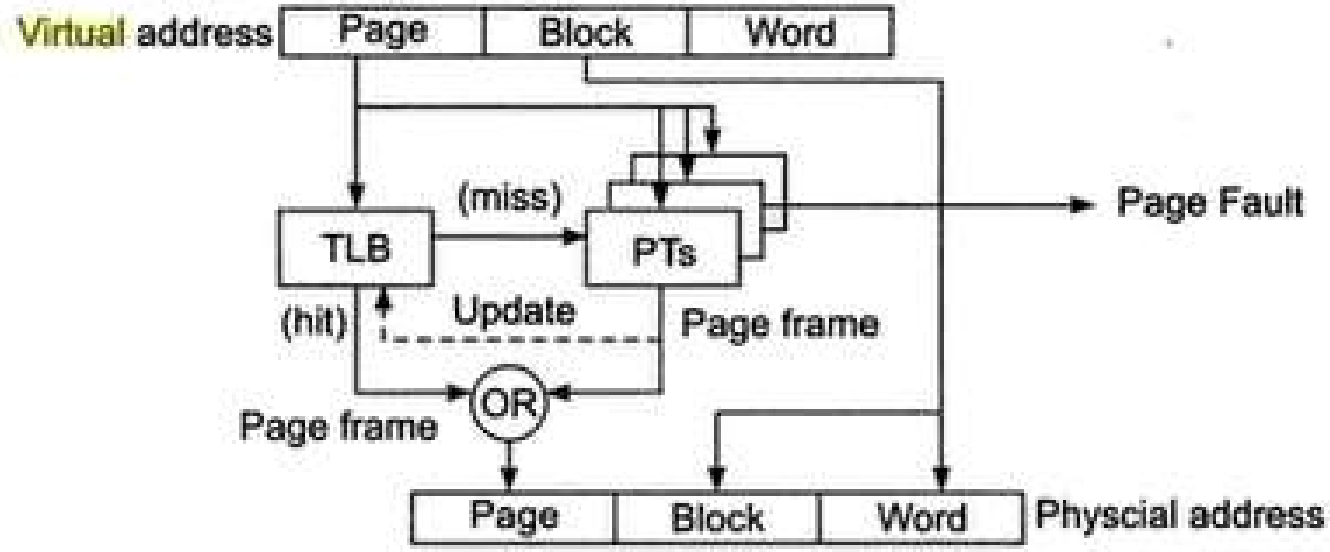


ADDRESS TRANSLATION MECHANISMS

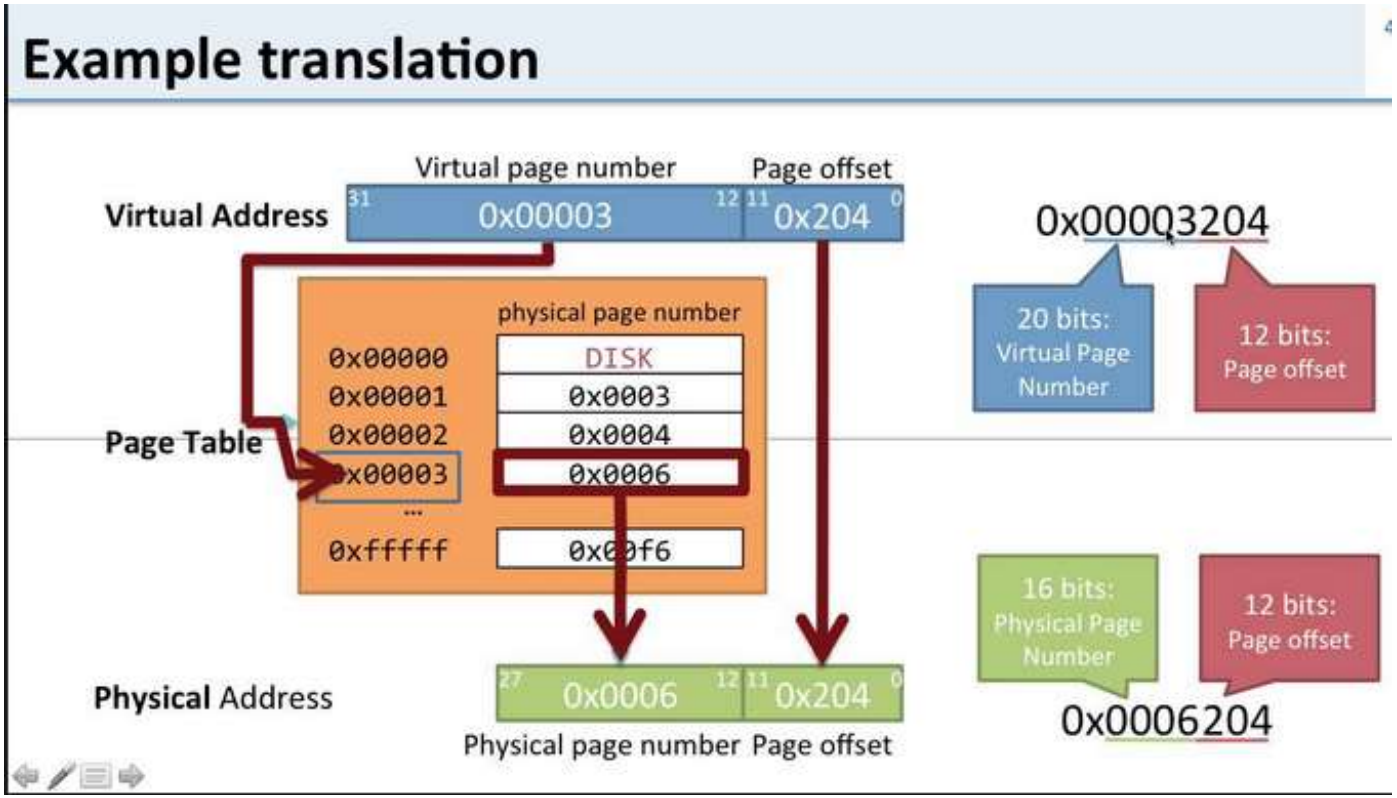
- [Virtual to physical] address translation requires use of a translation map.
  - The virtual address can be used with a **hash function** to locate the translation map (which is stored in the cache, an associative memory, or in main memory).
  - The translation map is comprised of a translation lookaside buffer, or TLB (usually in associative memory) and a page table (or tables). The virtual address is first sought in the TLB, and if that search succeeds, not further translation is necessary. Otherwise, the page table(s) must be referenced to obtain the translation result.
  - If the virtual address cannot be translated to a physical address because the required page is not present in primary memory, a *page fault* is reported.

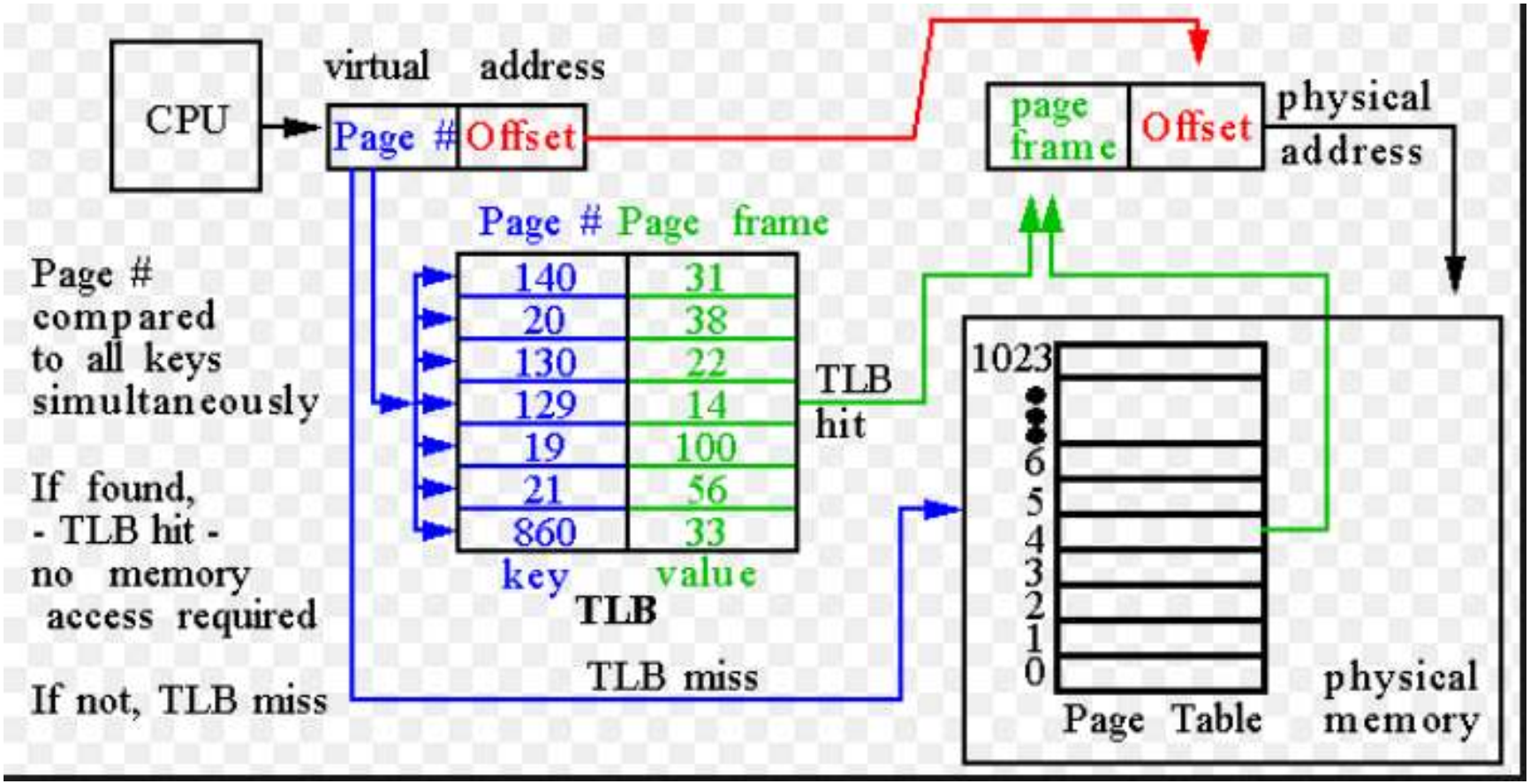


(a) Virtual addrress translation schemes (PT = page table)



(b) Use of a TLB and PTs for address translation


Example translation
Virtual page number
Page offset
Virtual Address
31
0x00003
12
11
0x204
0
0x00003204
20 bits:
Virtual Page
Number
12 bits:
Page offset
physical page number
Page Table
0x00000
0x00001
0x00002
0x00003
...
0xfffff
DISK
0x0003
0x0004
0x0006
0x00f6
16 bits:
Physical Page
Number
12 bits:
Page offset
Physical Address
27
0x0006
12
11
0x204
0
0x0006204
Physical page number
Page offset


CPU
virtual address
Page # Offset
page frame
Offset
physical address
Page # Page frame
Page # compared to all keys simultaneously
140 31
20 38
130 22
129 14
19 100
21 56
860 33
key value
TLB
TLB hit
If found, - TLB hit - no memory access required
If not, TLB miss
TLB miss
1023
6
5
4
3
2
1
0
Page Table
physical memory

Paged Memory Paging is a technique for partitioning both the physical memory and virtual memory into fixed-size pages. Exchange of information between them is conducted at the page level as described before. Page tables are used to map between pages and page frames. These tables are implemented in the main memory upon creation of user processes. Since many user processes may be created dynamically, the number of PTs maintained in the main memory can be very large. The *page table entries* (PTEs) are similar to the TLB entries, containing essentially (virtual page, page frame) address pairs.

Note that both TLB entries and PTEs need to be dynamically updated to reflect the latest memory reference history. Only "snapshots" of the history are maintained in these translation maps.

If the demanded page cannot be found in the PT, a *page fault* is declared. A page fault implies that the referenced page is not resident in the main memory. When a page fault occurs, the running process is suspended. A *context switch* is made to another ready-to-run process while the missing page is transferred from the disk or tape unit to the physical memory.

Segmented Memory A large number of pages can be shared by segmenting the virtual address space among multiple user programs simultaneously. A *segment* of scattered pages is formed logically in the virtual memory space. Segments are defined by users in order to declare a portion of the virtual address space.

In a *segmented memory system*, user programs can be logically structured as *segments*. Segments can invoke each other. Unlike pages, segments can have variable lengths. The management of a segmented memory system is much more complex due to the nonuniform segment size.

Segments are a user-oriented concept, providing logical structures of programs and data in the virtual address space. On the other hand, paging facilitates the management of physical memory. In a paged system, all page addresses form a linear address space within the virtual space.

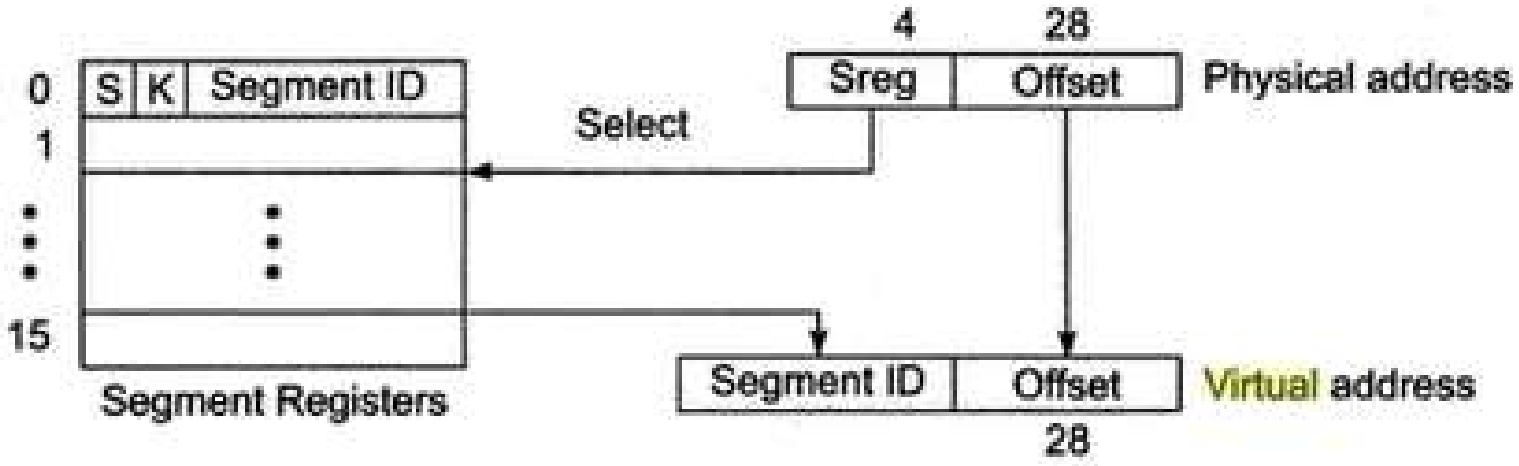
The segmented memory is arranged as a two-dimensional address space. Each virtual address in this space has a prefix field called the *segment number* and a postfix field called the offset within the segment. The *offset* addresses within each segment form one dimension of the contiguous addresses. The segment numbers, not necessarily contiguous to each other, form the second dimension of the address space.

Paged Segments The above two concepts of paging and segmentation can be combined to implement a type of virtual memory with *paged segments*. Within each segment, the addresses are divided into fixed-size pages. Each virtual address is thus divided into three fields. The upper field is the *segment number*, the middle one is the *page number*, and the lower one is the *offset* within each page.

Paged segments offer the advantages of both paged memory and segmented memory. For users, program files can be better logically structured. For the OS, the virtual memory can be systematically managed with fixed-size pages within each segment. Tradeoffs do exist among the sizes of the segment field, the page field, and the offset field. This sets limits on the number of segments that can be declared by users, the segment size (the number of pages within each segment), and the page size.

INVERTED PAGING

- A large virtual address space demands either large PTs or multilevel direct paging which will slow down the address translation process and thus lower the performance.
- Besides direct mapping, address translation maps can also be implemented with inverted mapping (Fig.4.21c).
  - An *inverted page table* is created for each page frame that has been allocated to users. Any virtual page number can be paired with a given physical page number.
  - Inverted page tables are accessed either by an associative search or by the use of a hashing function.
- Given a virtual address to be translated, the hardware searches the inverted PT for that address and, if it is found, uses the table index of the matching entry as the address of the desired page frame.
  - The size of an inverted PT is governed by the size of the physical space, while that of traditional PTs is determined by the size of the virtual space.
  - Because of limited physical space, no multiple levels are needed for the inverted page table.



(c) Inverted address mapping