



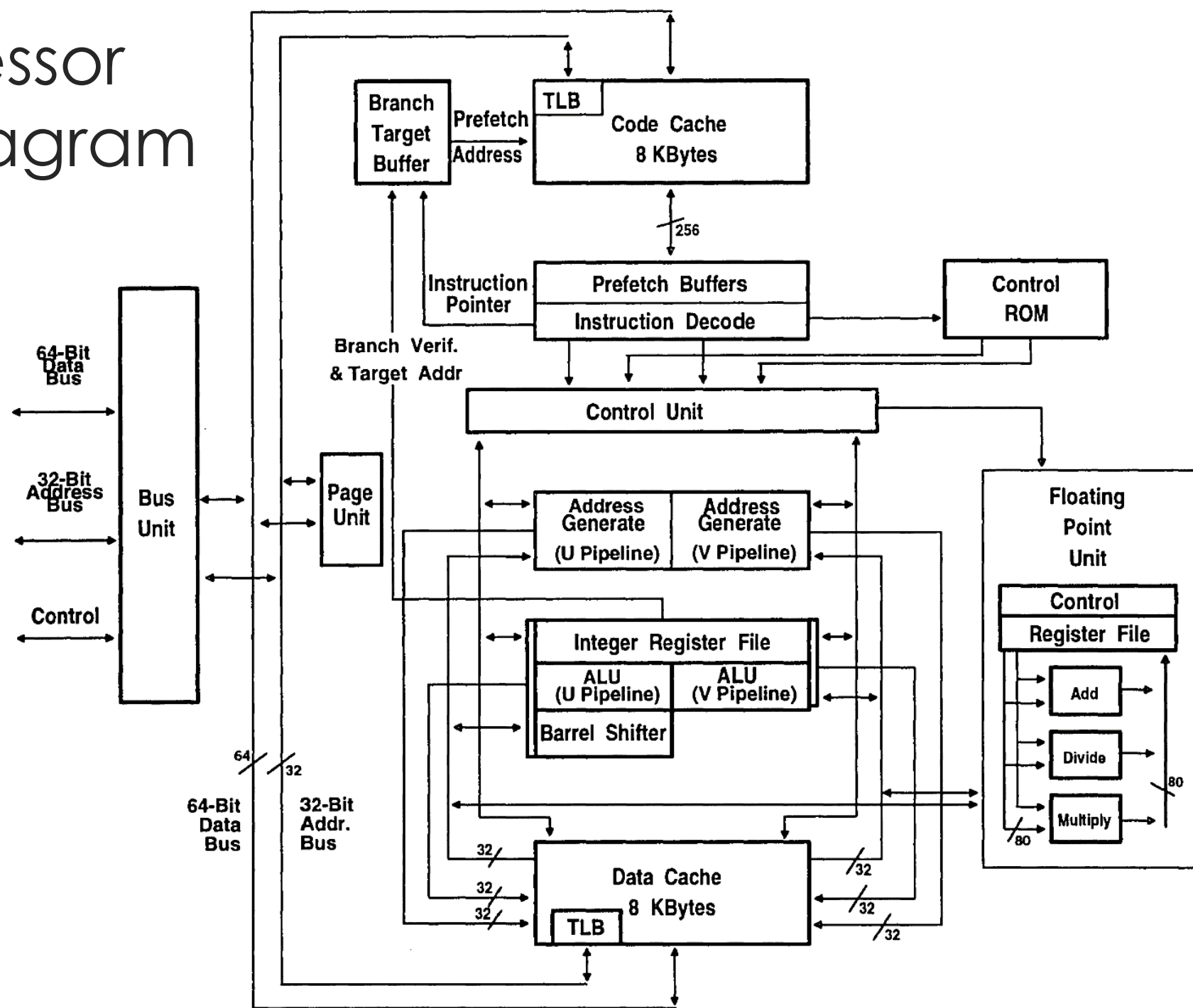
# CSC405 Microprocessor

## Pentium Processor

# Pentium Processor – Features

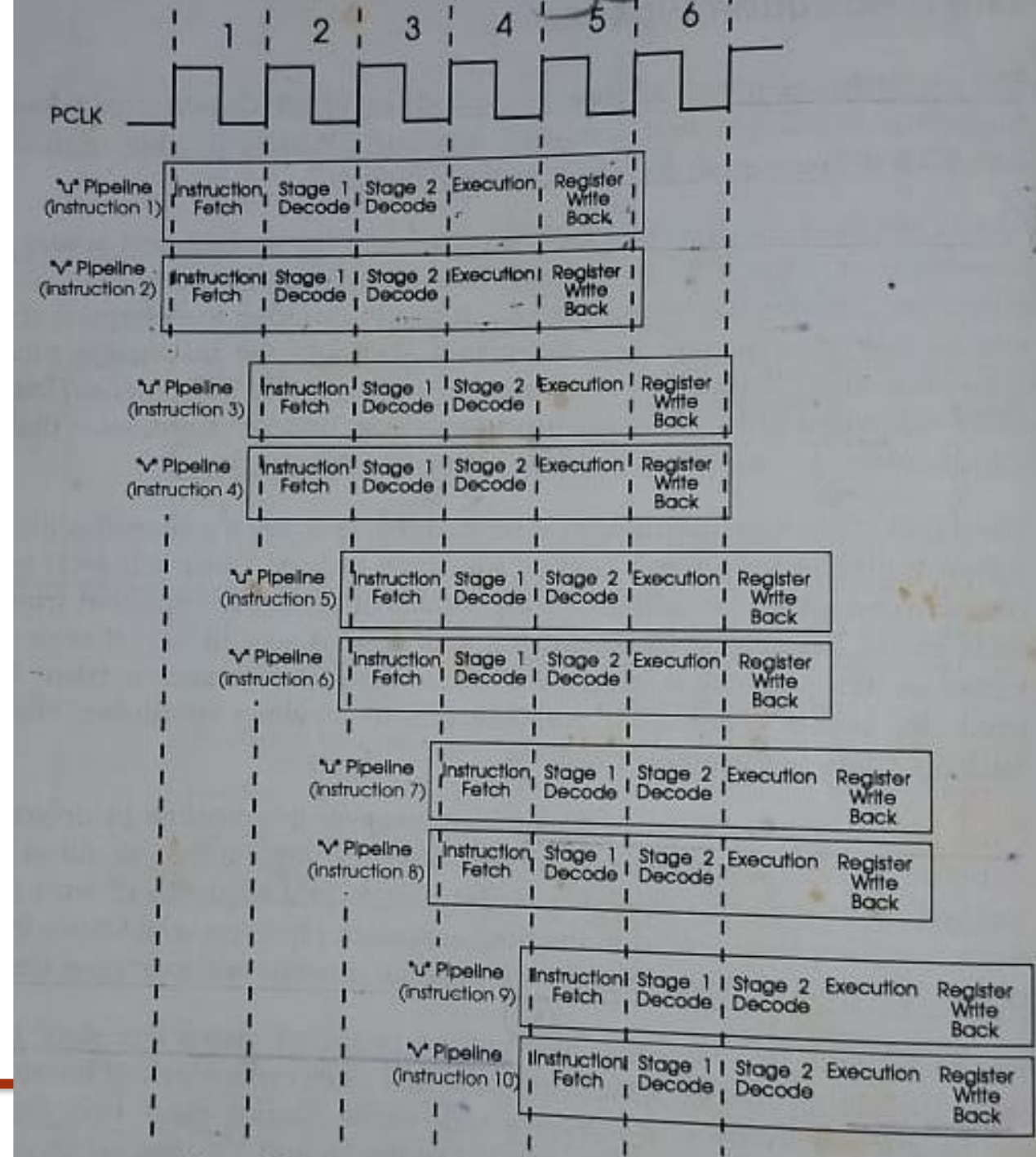
- **Wider data Bus : 64 bit data bus**
- **32 bit address bus**
- **Separate data and instruction caches (L1, 8 KB each)**
- **Parallel Integer Execution Units**
- **Floating point Unit**
- **Branch prediction**

# Pentium Processor Block Diagram



# Pentium@ Processor Pipe-line Execution

- PF Prefetch
- D1 Instruction Decode
- D2 Address Generate
- EX Execute - ALU and Cache Access
- WB Writeback



# Code Cache

- 2 way set associative cache
- 256 lines code cache and **prefetch buffer**, permitting prefetching of 32 bytes (256/8) of instructions
- Translation Lookaside Buffer (TLB) - translate linear addresses to the physical addresses

# Prefetcher



- Instructions are requested from code cache by the prefetcher
- If the requested line is not in cache, a burst bus cycle is run to external memory to perform a cache line fill

# Prefetch Buffers

- Four prefetch buffers within the processor works as two independent pairs.
  - When instructions are prefetched from cache, they are placed into one set of prefetch buffers.
  - The other set is used as when a branch operation is predicted.
- Prefetch buffer sends a pair of instructions to instruction decoder



# Branch Target Buffer

- Pentium uses Branch Target Buffer(BTB) for dynamic branch prediction.
- BTB is a special cache which stores the branch instruction that occur in the instruction stream.
- When branch operation is predicted, BTB request the predicted branch's target addresses from cache, which are placed in second pair of buffers that was previously idle.

# Instruction Decode Unit

- Instruction Decode occurs in two stages – Decode1 (D1) and Decode2(D2)
- During D1 – opcode is decoded in both pipelines
  - determine whether instructions can be paired
- D2 calculates the address of memory resident operands

# Control Unit

- This unit **interprets the instruction word** and **microcode entry point** fed to it by Instruction Decode Unit
- It handles exceptions, breakpoints and interrupts.
- It controls the integer pipelines and floating point sequences
- **Microcode ROM :**
  - Stores microcode sequences

# Address Generators



- Pentium provides two address generators(one for each pipeline).
- They generate the address specified by the instruction in their respective pipeline.

# Data Cache

- A separate internal Data Cache holds copies of the most frequently used data requested by the two integer pipelines and Floating Point Unit.
- The internal data cache is an 8KB write-back cache, organized as two-way set associative with 32 byte- lines.

# Paging Unit



- If paging is enabled ,the Paging Unit translates the linear address from address generator to a physical address.
- It can handle two linear addresses at the same time to support both pipelines.

# Arithmetic/Logic Units (ALUs)

- Two ALUs perform the arithmetic and logical operations specified by their instructions in their respective pipeline
- ALU for the “U” pipeline can complete the operation prior to ALU in the “V” pipeline but the opposite is not true.

# Floating Point Unit

- Perform floating point operations.
- It can accept up to two floating point operations per clock when one instruction is an exchange instruction.



# BUS UNIT



- It provides a physical interface between the Pentium Processor and the rest of the system.
- The Pentium communicates with the outside world via a 32-bit address bus and a 64-bit data bus.

# BUS UNIT

It consist of following units:

- Address Drivers and Receivers :
  - During bus cycles the address drivers push the address onto the processor's local address bus.
  - The address bus transfers addresses back to the Pentium address receivers during cache snoop cycles.
- Write Buffers:
  - The Pentium processor provides two write buffers ,one for each of the two internal execution pipelines.
  - This architecture improves performance when back-to-back writes occur.
- Data Bus Transceivers:
  - It consists of bidirectional tristate buffers for data.

# BUS UNIT

- Bus Master Control :
  - In multiprocessor system, request from the Bus Arbiter is handle by this unit.
- Bus Control Logic:
  - This unit generates control signals for the system bus.
- Level Two(L2)Cache Control:
  - To check whether L2 cache is present or not.
- Internal Cache Control:
  - Internal Cache control logic monitors input signal to determine when to snoop the address bus.
  - It also ensures proper cache coherency
- Parity Generation and Control:
  - To assure error free transmission of data

# Integer Pipeline Stages

- In Pentium, there are two instruction pipelines U Pipeline and V pipeline.
- These are five stage pipelines and operate independently.
  - PF : Prefetch
  - D1 : Instruction Decode
  - D2 : Address generate
  - EX : Execute, Cache and ALU Access
  - WB : Write back

# Integer Pipeline Stages

## **Prefetch(PF) :**

- Instructions are prefetched from the instruction cache or memory and fed into the PF stage of both U and V Pipeline.

## **Decode1(D1):**

- In this stage, decoder in each pipeline checks if the current pair of instructions can execute together.
- If the instruction contains a prefix byte ,an additional clock cycle is required in this stage. Also such an instruction may only execute in the U pipeline and may not be paired with any other instruction.
- It decodes the instruction to generate a control word
- A single control word causes direct execution of an instruction
- Complex instructions require microcode control sequencing

# Integer Pipeline Stages

## Address Generate Stage(D2):

- Decodes the control word
- Address of memory resident operands are calculated

## Execute (EX):

- ALU operation
- **Data** cache is **accessed** at this stage
- For **both** ALU and data cache access require **more than one clock**.

## Write back(WB):

- The CPU **stores the result and updates the flags**

# Integer Instruction Pairing Rules

In order to issue two instructions simultaneously they must satisfy the following conditions:

- Both instructions in the pair must be "simple"
- There must be no read-after-write or write-after-write register dependencies between them
  - Register Contention (read-after-write)
    - `mov ax, 4b`
    - `mov [bp], ax`

# Integer Instruction Pairing Rules

- Register Contention (write-after-write)
  - `Mov ax, 4b`
  - `Mov ax, [bp]`
- Neither instruction may contain both a displacement and an immediate
- Implicit Register contention
  - If two instructions imply reference to the same register
- Exceptions
  - Flag References - compare and branch
  - Stack Pointer Reference - pushes or pops



# Integer Instruction Pairing Rules

## Simple Instructions

- They are entirely hardwired
- They do not require any microcode control
- Executes in one clock cycle
  - Exception: ALU mem, reg and ALU reg, mem are 3 and 2 clock operations respectively (Arithmetic and logic instructions that use both register and memory operand)

# Integer Instruction Pairing Rules

- integer instructions are considered simple and may be paired

1. mov reg, reg/mem/imm
2. mov mem, reg/imm
3. alu reg, reg/mem/imm
4. alu mem, reg/imm
5. inc reg/mem
6. dec reg/mem
7. push reg/mem
8. pop reg
9. Lea reg,mem
10. jmp/call/jcc near
11. nop

# Instruction Issue Algorithm

- Decode the two consecutive instructions I1 and I2
- If the following are all true
  - I1 and I2 are simple instructions
  - I1 is not a jump instruction
  - Destination of I1 is not a source of I2
  - Destination of I1 is not a destination of I2
- Then issue I1 to u pipeline and I2 to v pipeline
- Else issue I1 to u pipeline

# Floating Point Pipeline

- The floating point pipeline has 8 stages as follows:

## 1. Prefetch(PF) :

- Instructions are prefetched **from** the on-chip **instruction cache**

## 2. Instruction Decode(D1):

- Two parallel decoders attempt to decode and issue the next two sequential instructions
- It decodes the instruction to **generate a control word**
- A single control word causes direct execution of an instruction
- Complex instructions require microcode control sequencing

# Floating Point Pipeline

## 3. Address Generate (D2):

- Decodes the control word
- **Address** of memory resident **operands** are calculated

## 4. Memory and Register Read (Execution Stage) (EX):

- **Register read, memory read or memory write** performed as required by the instruction to access an operand.

## 5. Floating Point Execution Stage 1(X1):

- Information from register or memory is written into FP register.
- **Data is converted to floating point format** before being loaded into the floating point unit

# Floating Point Pipeline

## 6. Floating Point Execution Stage 2(X2):

- Floating point **operation performed** within floating point unit.

## 7. Write FP Result (WF):

- Floating point results are **rounded and** the result is **written to** the **target floating point register**.

## 8. Error Reporting(ER)

- If an error is detected, an error reporting stage is entered where the **error is reported** and **FPU status word is updated**

# Instruction Issue for Floating Point Unit

- The rules of how floating-point (FP) instructions get issued on the Pentium processor are :

1. When a pair of FP instructions is issued to the FPU, **only the FXCH instruction can be the second** instruction of the pair.

The **first** instruction of the pair must be one of a set F where **F = [FLD, FADD, FSUB, FMUL, FDIV, FCOM, FUCOM, FTST, FABS, FCHS]**.

2. **FP instructions other than FXCH and instructions belonging to set F,** always get issued **singly** to the FPU.
3. FP instructions that are **not directly followed by an FXCH** instruction are issued **singly** to the FPU.

# Branch Prediction Logic

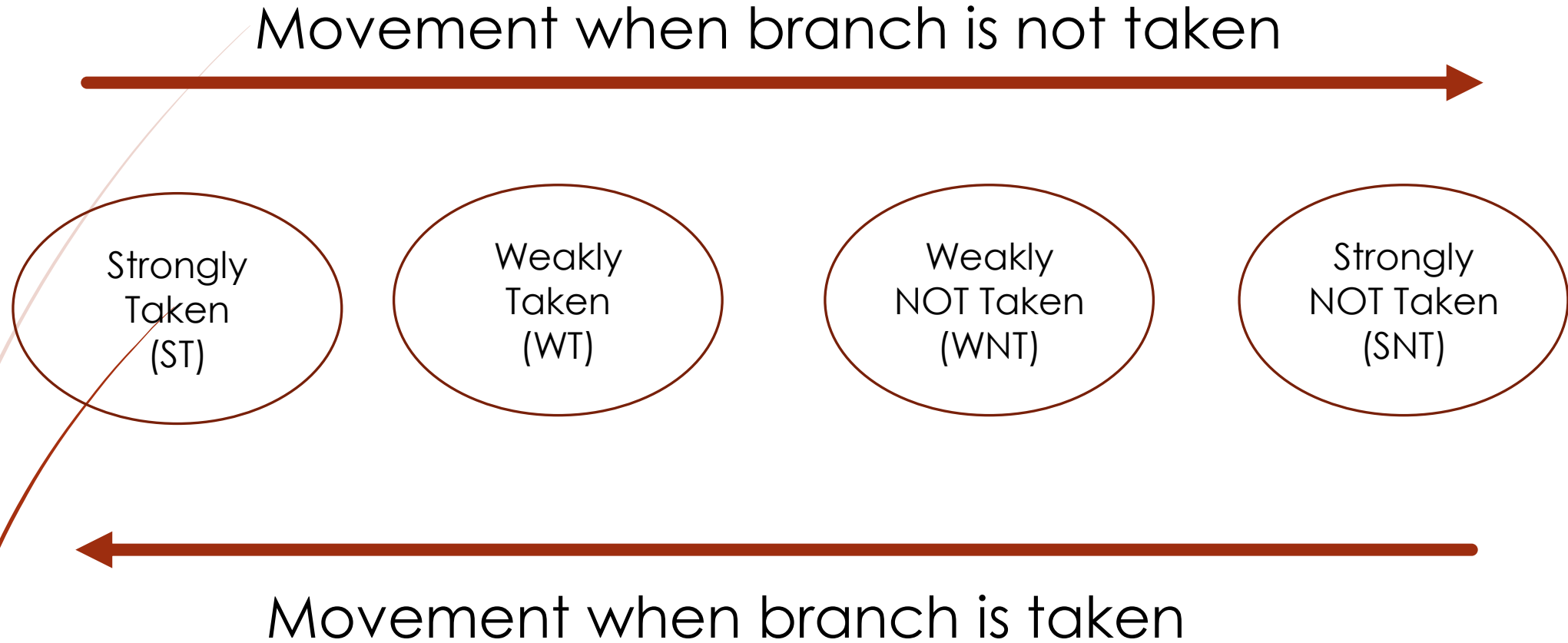
- Other than the Superscalar ability of the Pentium processor, the branch prediction mechanism is a much improvement.
- Predicting the behaviors of branches can have a very strong impact on the performance of a machine. Since a wrong prediction would result in a flush of the pipes and wasted cycles.
- The branch prediction mechanism is done through a branch target buffer. The branch target buffer contains the information about all branches



# Branch Prediction Logic

- BTB
  - 4 way set associative cache with 256 entries
- Directory entry of each line
  - A valid bit
  - History Bits
  - Source memory address
- Target address of the branch is stored in the corresponding entry in BTB

# Branch Target Buffer Branch History Bits

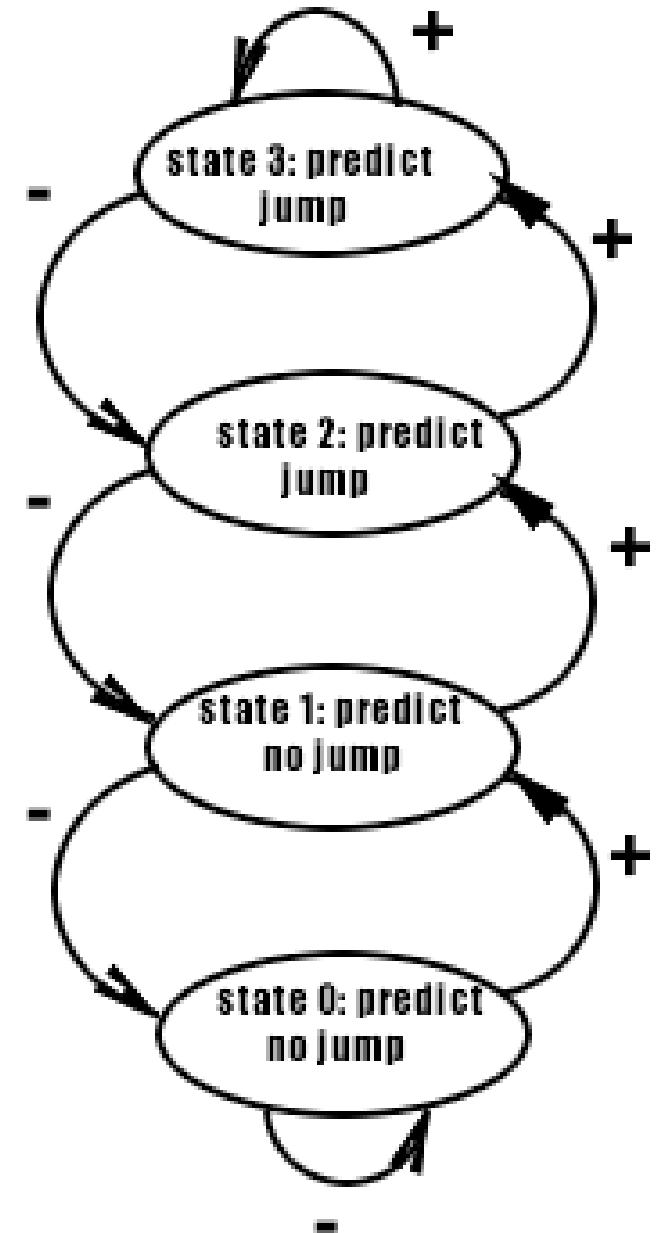


# Branch Prediction Logic

- The prediction of whether a jump will occur or no, is based on the branch's previous behavior.
- There are four possible states that depict a branch's disposition to jump:
  - Stage 0: Very unlikely a jump will occur
  - Stage 1: Unlikely a jump will occur
  - Stage 2: Likely a jump will occur
  - Stage 3: Very likely a jump will occur

# Branch Prediction Logic

- When a branch has its address in the branch target buffer, its behavior is tracked.
- This diagram portrays the four stages associated branch prediction.
- If a branch doesn't jump two times in a row, it will go down to State 0.
- Once in Stage 0, the algorithm won't predict another jump unless the branch will jump for two consecutive jumps (so it will go from State 0 to State 2)
- Once in Stage 3, the algorithm won't predict another no jump unless the branch is not taken for two consecutive times.



# Data Cache Consistency Protocol (MESI Protocol)

| State     | Description   |
|-----------|---|
| Modified  | The line in cache has been updated due to a write hit in the cache. This alerts the cache subsystem to snoop the system bus and write the modified line back to memory when a snoop hit to the line is detected |
| Exclusive | Indicates that this cache knows that no other cache in the system possesses a copy of this line therefore it is exclusive to this line  |
| Shared    | Indicates that the line may be present in several caches and if so that an exact duplicate of the information exists in each source   |
| Invalid   | The initial state after reset indicating that the line is not present in the cache  |

# Data Cache Consistency Protocol (MESI Protocol)

