Trace Cache:

Trace cache keeps the trace of the data being executed. Trace cache can be called as shadow cache. Trace cache is placed with execution engine.

It is the responsibility of fetch unit to ~~keep~~ fill the trace cache.

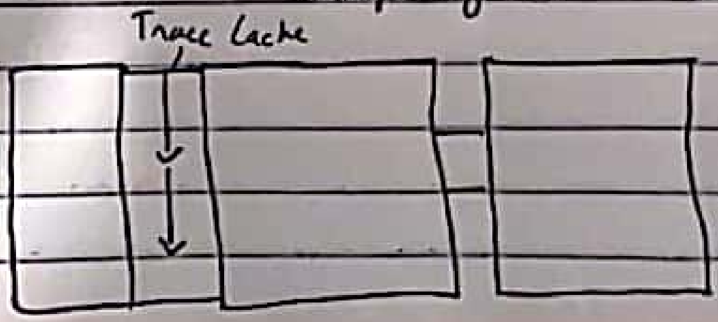
. When instructions are in Trace Cache they are called properly decoded control words.

When instruction is decoded completely it brokes down to its microoperations.

Steps:

- Instruction is decoded into its microoperations (control words)
- Microoperation fusion is taken place

We fuse these microops together — $\left[\begin{array}{l} 1 \quad I1 \leftarrow m[ar],\ Pc \leftarrow Pc+1 \\ 1 \quad e \leftarrow 0 \end{array}\right.$

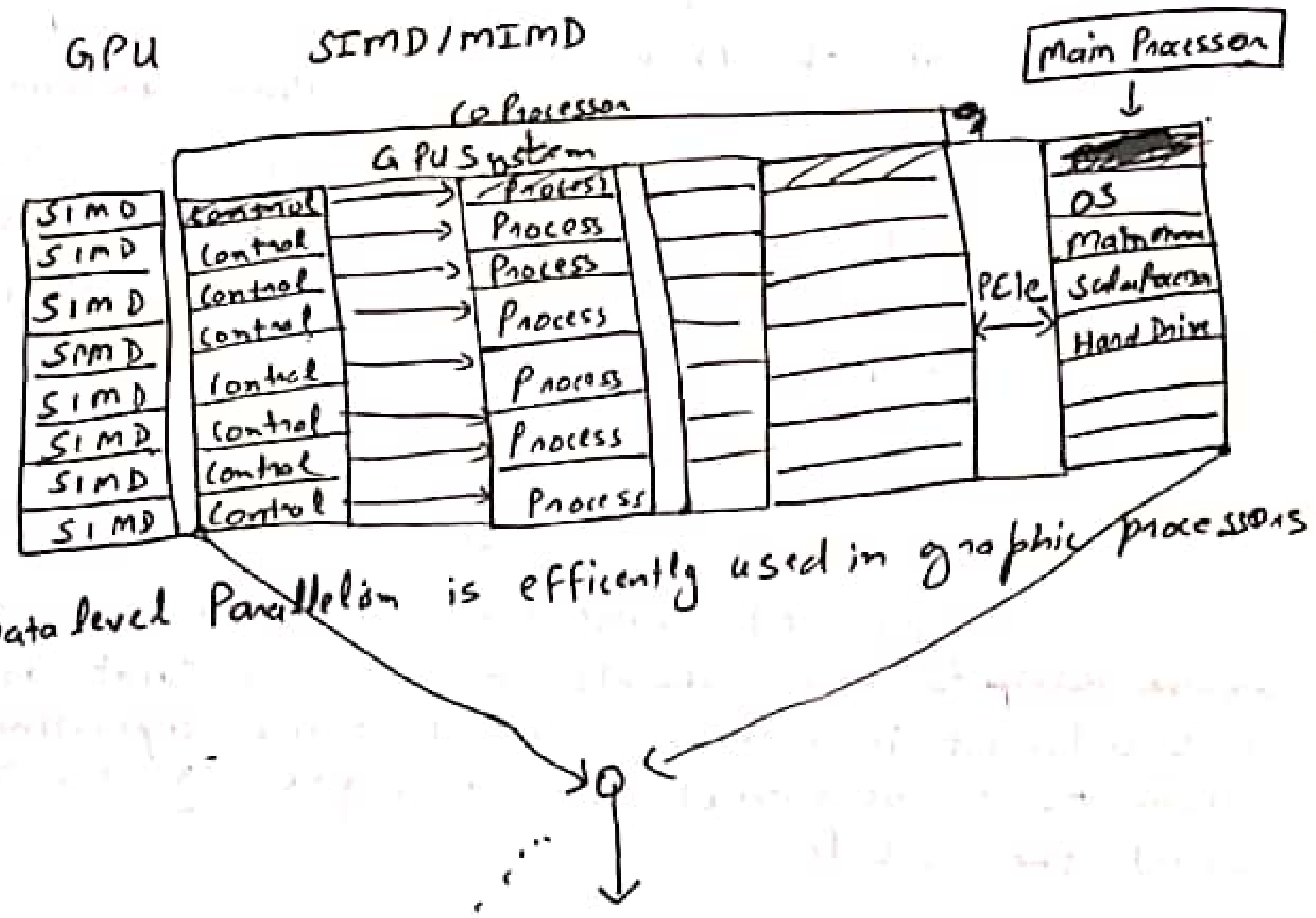


29-01-21

GPU (Graphic Processors)

- It is twodimensional architecture to process two dimensional data-

GPU SIMD/MIMD



Data level Parallelism is efficently used in graphic processors

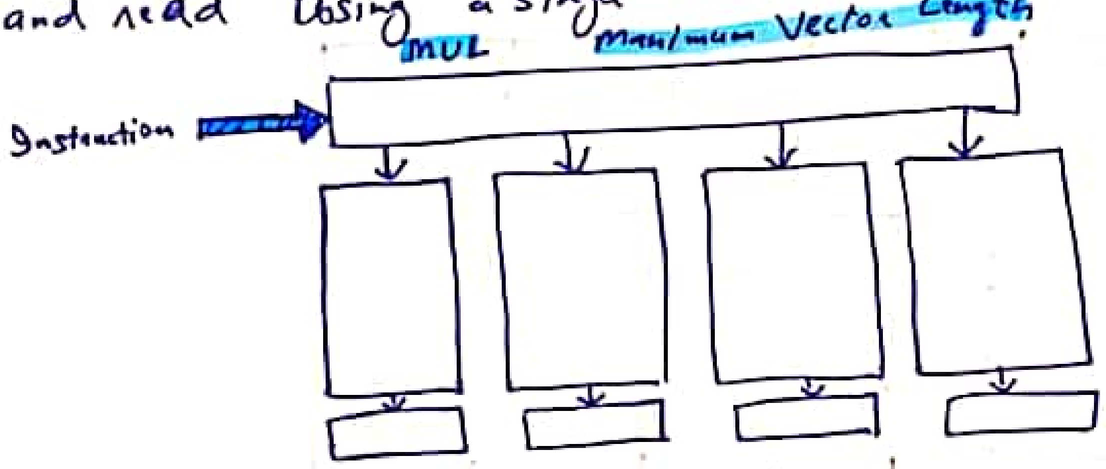
Now this diagram shows CUDA
:-
- Instruction set in Graphic Processor is small
But
- Data set in Graphic Processor is very large

22-01-21

Vector Processors: It is from SIMD ~~[illegible]~~ class

Streaming Instructions: (Floating Point) SSE
mmx Unit (Integer)
AVX Unit (Advance Vector Extension)

Vectors are of very large length. They are load, stored and read using a single command.



Stride:-

$C = A * B$

Metrices

(Row major, Column Major Concept)

we can read elements with stride

A[10] [10]		B[10] [10]	
A[0] [0]	1000	B[0] [0]	2000
A[0][1]	1001	B[0] [1]	2001

Gather-Scatter :-

Sparse Matrix

A [M[I]]

MMX Technology

Intel MMX technology is used to run the multimedia and communication applications. It includes new instruction and data types that permit to gain a new level of performance.

MMX technology is the most significant enhancement to intel 1386™ processor. Which extended the architecture to 32 bit. Processor with MMX technology will deliver significant performance for multimedia and communication applications.

Data Types:

The principal data type of the MMX instruction set is the packed, fixed point integer where multiple integer words combined together into a single 64bit- quantity. The 64-bit quantity is moved towards into 64 bit MMX register. The four MMX technology data types are:

1. Packed Byte
2. Packed word
3. Packed double word
4. Quad word

Instructions:

The MMX instructions cover several functional areas including:

1. Basic arithmetic operations such as add,subtract,multiply,althemetic shift and multiply –add
2. Comparison operations
3. Conversion instructions to convert between the new data types -pack data together ,and unpack from small to larger data types
4. Logical operations such as AND, AND NOT, OR, and XOR.
5. Shift operations
6. Data transfer (MOV) instruction for MMX register –to-register transfers, or 64-bit and 32 bit load/store to memory.

Streaming SMID Extension (SSE)

Instructions that operate on scalar or vector (packed) integer and floating point numbers.Instrcution set comprises the legacy SSE and extended SSE instruction sets.

1. SSE1
2. SSE2
3. SSSE3

Legacy SSE architecture supports operations Involving 128-bit vectors and explore the base programing model including the SSE registers, the Media extension Control and status register (MXCSR), and the instruction exception behavior.

The streaming SIMD Extension (SSE) instruction set is extended to include

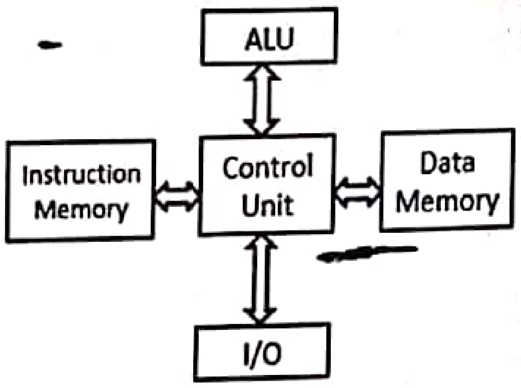
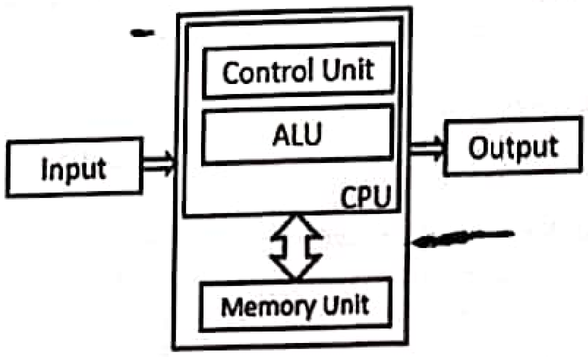
1. AVX
2. FMA
3. FMA4
4. XOP

A significant feature of the SSE is the doubling of the XMM register. This type of registers are referred to as the YMM registers. The SSE instructions can be used in legacy mode or long (64-bit) mode.

Following Advantages, Compilation of 64-bit mode:

1. Access to an additional eight YMM/XMM registers for a total of 16
2. Access to an additional eight 64-bit general –purpose registers for a total of 16
3. Access to the 64-bit virtual address space and the RIP –relative addressing mode.

Difference between Von Neumann and Harvard Architecture

Point of Comparison	Harvard Architecture	Von Neumann Architecture
rrangement	In Harvard architecture, the CPU is connected with both the data memory (RAM) and program memory (ROM), separately.  ALU Instruction Memory Control Unit Data Memory I/O Harvard Model	In Von-Neumann architecture, there is no separate data and program memory. Instead, a single memory connection is given to the CPU.  Control Unit ALU CPU Input Output Memory Unit Von Neumann Model
ardware equirement	It requires more hardware since it will be requiring separate data and address bus for each memory.	In contrast to the Harvard architecture, this requires less hardware since only a common memory needs to be reached.
pace equirement	This requires more space.	Von-Neumann Architecture requires less space.
peed of xecution	Speed of execution is faster because the processor fetches data and instructions simultaneously .	Speed of execution is slower since it cannot fetch the data and instructions at the same time.
pace usage	It results in wastage of space since if the space is left in the data memory then the instructions memory cannot use the space of the data memory and vice-versa.	Space is not wasted because the space of the data memory can be utilized by the instructions memory and vice-versa.

ontrolling	Controlling becomes complex since data and instructions are to be fetched simultaneously.	Controlling becomes simpler since either data or instructions are to be fetched at a time.
ost	Comparatively high cost.	It is cheaper.
erformance	Easier to pipeline, so high performance can be achieve.	Low performance as compared to Harvard architecture.
ycle per nstruction	Processor can complete an instruction in one cycle	Processor needs two clock cycles to complete an instruction.
us	Harvard architecture is required separate bus for instruction and data.	Von Neumann architecture is required only one bus for instruction and data.
omplexity	complicated	simple
nemo[illegible]	It required two memories for their instruction and data.	It required only one memory for their instruction and data.

What is Von Neumann Architecture?
It's a theoretical design based on the concept of stored-program computers where program data and instruction data are stored in the same memory.

The architecture was designed by the renowned mathematician and physicist John Von Neumann in 1945. Until the Von Neumann concept of computer design, computing machines were designed for a single predetermined purpose that would lack sophistication because of the manual rewiring of circuitry.

The idea behind the Von Neumann architectures is the ability to store instructions in the memory along with the data on which the instructions operate. In short, the Von Neumann architecture refers to a general framework that a computer's hardware, programming, and data should follow.

The Von Neumann architecture consists of three distinct components: a central processing unit (CPU), memory unit, and input/output (I/O) interfaces. The CPU is the heart of the computer system that consists of three main components: the Arithmetic and Logic Unit (ALU), the control unit (CU), and registers.

The ALU is responsible for carrying out all arithmetic and logic operations on data, whereas the control unit determines the order of flow of instructions that need to be executed in programs by issuing control signals to the hardware.

The registers are basically temporary storage locations that store addresses of the instructions that need to be executed. The memory unit consist of RAM, which is the main memory used to store program data and instructions. The I/O interfaces allows the users to communicate with the outside world such as storage devices.

What is Harvard Architecture?
It is a computer architecture with physically separate storage and signal pathways for program data and instructions. Unlike Von Neumann architecture which employs a single bus to both fetch instructions from memory and transfer data from one part of a computer to another, Harvard architecture has separate memory space for data and instruction.

Both the concepts are similar except the way they access memories. The idea behind the Harvard architecture is to split the memory into two parts – one for data and another for programs. The terms was based on the original Harvard Mark I relay based computer which employed a system that would allow both data and transfers and instruction fetches to be performed at the same time.

Real world computer designs are actually based on modified Harvard architecture and are commonly used in microcontrollers and DSP (Digital Signal Processing).

Difference between Von Neumann and Harvard Architecture

Lesson 3 Pipelining

Pipelining

Pipelining is crucial to improving performance in processors it increases throughput and reduces cycle time. The downside of pipelines are the increase in hazards, both control and data.

Pipelining in a Processor

[illegible]

Five stages in a basic pipeline:
Fetch, Read/Decode, ALU, Memory Access, Write the Registers

~~Pipelin~~ing the instructions takes the same amount of ~~time, but~~ throughput is improved.

Pipelining CPI

If there an instruction has to wait at a pipeline stage, all the instructions ahead of it proceed through the pipeline, all the instructions behind it are also stalled. This is called a delay in the pipeline. The pipeline ahead of the delay will not have instructions to execute as the pipeline empties and the instructions behind the delay will be stalled.

As the number of delays increase through the pipeline, the CPI will increase.

Pipeline Stalls and Flushes

Pipeline Flush: Branches can cause bubbles when the incorrect branch is taken. When this happens all the incorrect instructions that were fetched must be flushed from the pipeline and replaced with NOPs. Then the correct instructions must be fetched.

Control Dependencies

Control dependencies: When an instruction is dependent on the outcome of a branch decision, these instructions are said to have a control dependence.

20% of instructions are branches and jumps
50% of branch and jump instructions are taken

Overall CPI = CPI of program + % of instructions mispredicted * penalty for misprediction

Data Dependencies

Data dependence: When an instruction needs data from another instruction that is called a data dependence.

Type of data dependencies:

1. RAW - read after write. Also called Flow, True Dependence.

The following dependencies are False or Name dependencies.

2. WAW - write after write. Also called Output Dependence.
3. WAR - write after read. Also called Anti-Dependence

RAR -read after read is not a dependence.

Dependencies and Hazards

Dependencies are caused by the program, not the pipeline.
Some dependences will not cause problems, but some, like RAW, can cause problems in pipelines.
Hazards are true dependencies that result in incorrect execution of the program,but not all true dependencies will result in a hazard. Hazards are caused by both the program and the pipeline.

Handling of Hazards

First step to handling hazards: Detect only those dependencies that will cause hazards.
Second Step to handling hazards: Remove the hazard.

Options for removing hazards are:

1. Flush dependent instructions out of the pipeline.
   This method is used with control dependencies. Since the wrong value is in the pipeline it needs to be flushed.
2. Stall dependent instructions in the pipeline.
   Th~~is method~~ is used for data dependencies. This will give ~~the~~ data time to get written to registers before it is needed by a later instruction.
3. Fix the values read by dependent instructions.
   This method is also used for data dependencies. Instead of stalling the instruction until the correct values are in the registers, the pipeline can forward the required values. The values from an ALU for example can be used as soon as they computed, rather than waiting.
   This method does not always work.

How Many Stages

Every pipeline should be achieving the required CPI, it is different for every pipeline.
When more stages are added to a pipeline:

1. There are more hazards introduced into the pipeline.
2. The penalty for hazards increases.
3. There is less work for each stage, so the cycle time can be smaller.

Lesson 11 VLIW

VLIW

VLIW processors are another way to improve performance. These processors work best with regular tasks- such as loops and array manipulations.

More than 1 IPC

Processors that can issue more than 1 instruction per cycle:

-Out of Order Superscalar

-It can issue multiple instructions per cycle
-It can look at a lot of instructions at a time for scheduling
-Very expensive - with many reservation stations, etc.
- A compiler can help with improving IPC

-In Order Superscalar

-It can issue multiple instructions per cycle
-It can look at fewer instructions at a time for scheduling than OOO processor
-It is less expensive that OOO processor
-It needs help from a compiler to improve IPC

-Very Long Instruction Word (VLIW)

-It executes 1 big instruction per cycle
-It does not do instruction scheduling, it just executes the next large instruction
-It is the least expensive of the three listed
-It really requires a good compiler

Superscalar Vs. VLIW

A superscalar processor:

1. Gets multiple instructions
2. Checks for dependencies
3. Then sends instructions to the execution units for parallel execution when it can.

A VLIW Processor:

1. The compiler looks for dependencies
2. If there are dependencies it loads them into separate instruction words. This can lead to much larger number of bytes for a program in VLIW.

VLIW: The Good and the Bad Good:

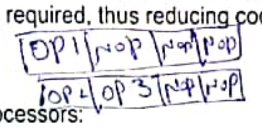
-The compiler does the work and this program is run over and over. Thus, the compiler can take the time to find good instruction scheduling.
-The hardware is simpler than for Superscalar
-It can be energy efficient
-It works well on "regular code" such as loops and arrays.

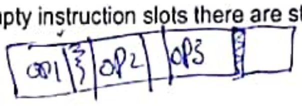
Bad:

-Latencies of instructions are not always the same -Many applications are irregular -Code bloat

VLIW Instructions

-VLIW instructions have all the usual ISA opcodes -Fully support predication
-Require many registers because of the scheduling optimizations
-Branch hints because the compiler needs to tell the hardware its predictions
-VLIW instruction compaction - instead of using NOPs for empty instruction slots there are stops. This reduces the number of instructions required, thus reducing code bloat.





VLIW Examples

Examples of VLIW processors:

Itanium Processor - too complicated, not good with irregular code
DSP Processors - usually have excellent performance and energy efficient

7 Types of Instruction Set

Reduced Instruction Set Computer (RISC)

Reduced Instruction Set Computer (RISC) is an instruction set architecture (ISA) which has fewer cycles per instruction (CPI) than a complex instruction set computer (CISC).

RISC processors are also used in supercomputers such as Summit, which, as of November 2018, is the world's fastest supercomputer as ranked by the TOP500 project.

Features of RISC.
↳ one inst per clock period.
↳ All inst have same size
↳ CPU access memory only for load & store operations
↳ simple & few addressing modes.

Features of CISC.
↳ more work per instruction
↳ wide variety of addressing modes.
↳ variable instruction lengths & execution time per instruction.
↳ CISC machines attempt to reduce the "semantic gap"

[illegible]

Complex Instruction Set Computer (CISC)

Complex Instruction Set Computer (CISC) is an instruction set architecture (ISA) which has fewer instructions per program than a Reduced instruction set computer (RISC).

[illegible] clock Period, T, Can not be reduced beyond a certain limit.

Minimal instruction set computers (MISC)

Minimal instruction set computers (MISC) is a processor architecture with a very small number of basic instruction operations and corresponding opcodes.

complex addressing modes delay operand fetch from memory.

As a result of this is a smaller instruction set, a smaller and faster instruction set decode unit, and faster operation of individual instructions. The disadvantage is that smaller instruction set always have more sequential dependencies, reducing instruction-level parallelism.

Difficult to make efficient use of speedup techniques.

Very long instruction word (VLIW)

Very long instruction word (VLIW) is an instruction set architectures **designed to exploit instruction level parallelism** (ILP).
Central processing units (CPU, processor) mostly allow programs to specify instructions to execute in sequence only, a VLIW processor allows programs to explicitly specify instructions to execute in parallel. This design is intended to allow higher performance without the complexity inherent in some other designs.

Explicitly parallel instruction computing (EPIC)

Explicitly parallel instruction computing (EPIC) is an instruction set that permits microprocessors **to execute software instructions in parallel by using the compiler**, rather than complex on-die circuitry, to control parallel instruction execution.
This was intended to allow simple performance scaling without resorting to higher clock frequencies.

One instruction set computer (OISC)

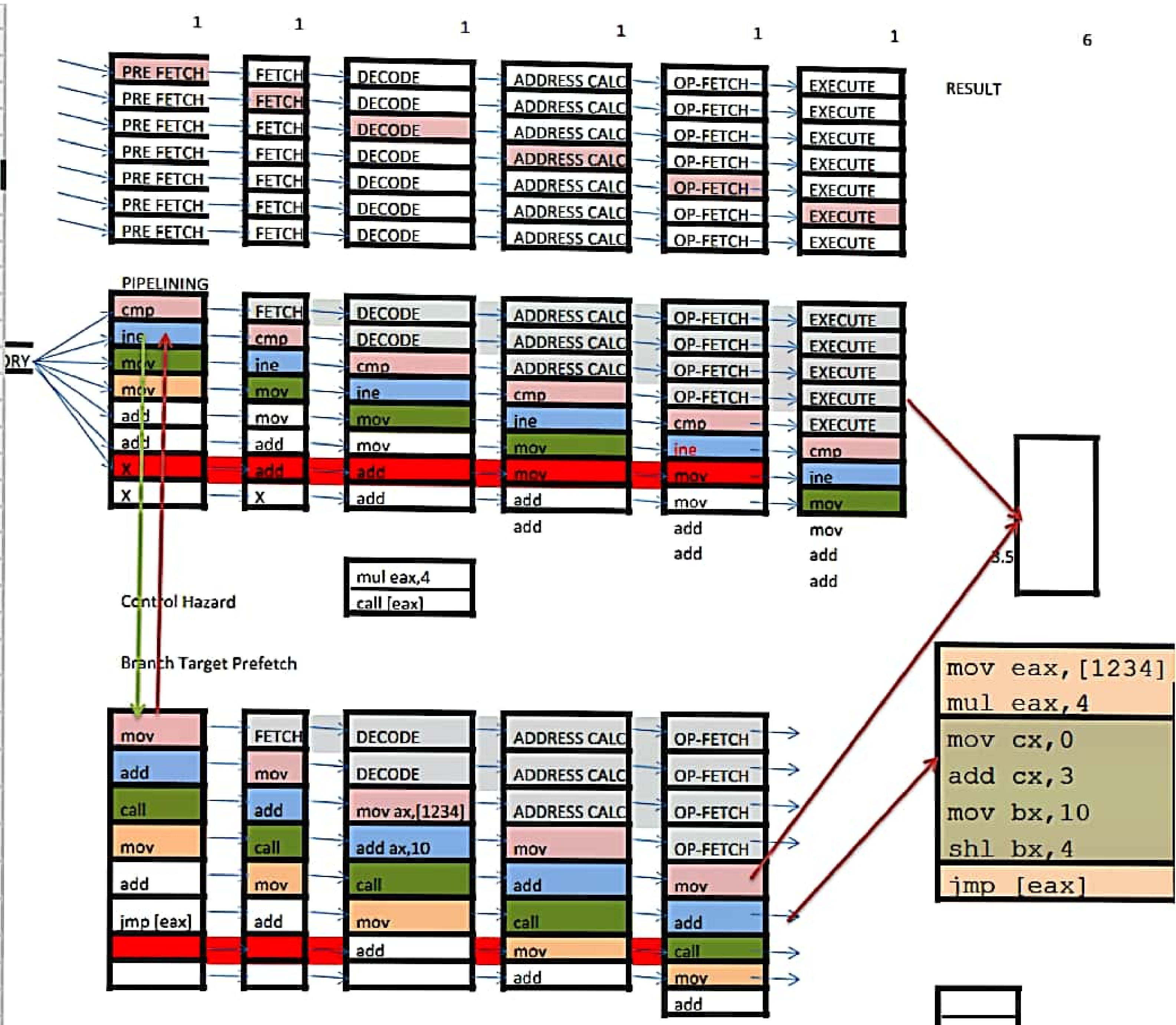
One instruction set computer (OISC) is an abstract machine that uses only one instruction obviating the need for a machine language opcode.

OISCs have been recommended as guides in teaching computer architecture and have been used as computational models in structural computing research.

Zero instruction set computer (ZISC)

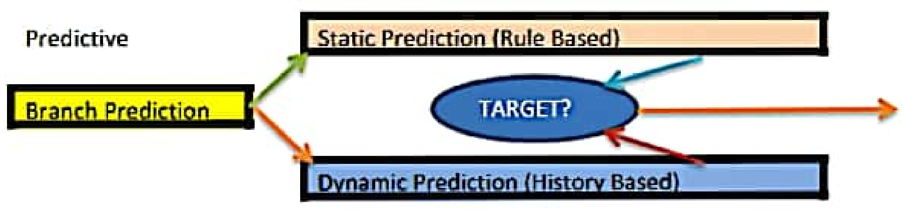
Zero instruction set computer (ZISC) is a computer architecture based on pattern matching and absence of (micro-)instructions in the classical sense.

These chips are known for being thought of as comparable to the neural networks being marketed for the number of "synapses" and "neurons"



```
mov eax,[1234]
mul eax,4
mov cx,0
add cx,3
mov bx,10
shl bx,4
jmp [eax]
```

"03214438334


Predictive
Static Prediction (Rule Based)
Branch Prediction
TARGET?
Dynamic Prediction (History Based)

Code Driven Processors	It will execute the instruction for which the CODE is available
Data Driven Processors	It will execute the instruction for which the DATA is available

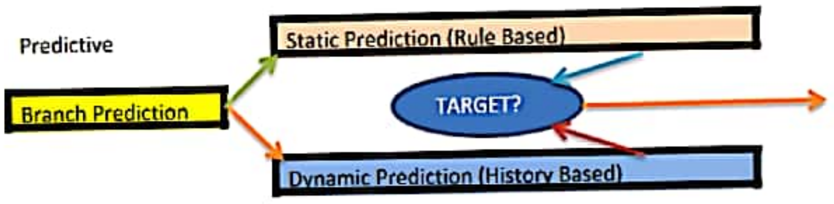
Dependency

Out of Order Execution unit

Data Hazard

Scanned with CamScanner

Predictive



Branch Prediction
Static Prediction (Rule Based)
TARGET?
Dynamic Prediction (History Based)

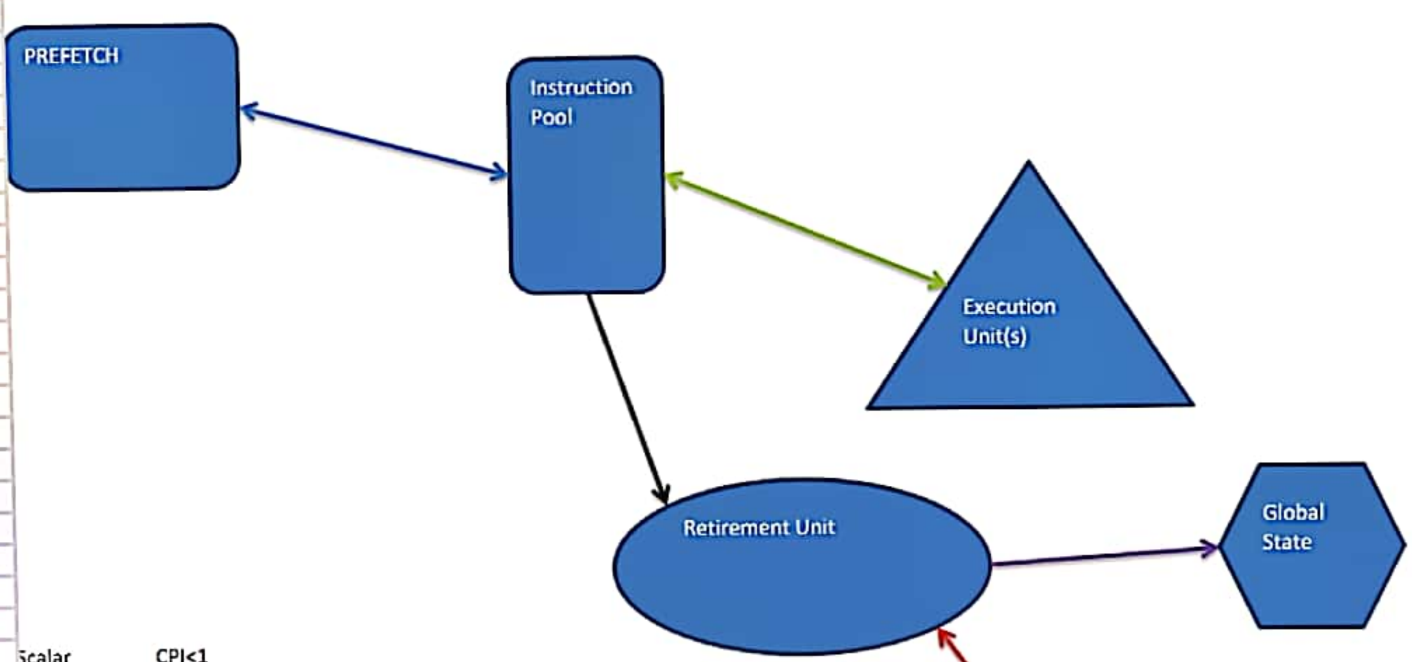
"03214438334

Code Driven Processors	It will execute the instruction for which the CODE is av
Data Driven Processors	It will execute the instruction for which the DATA is av

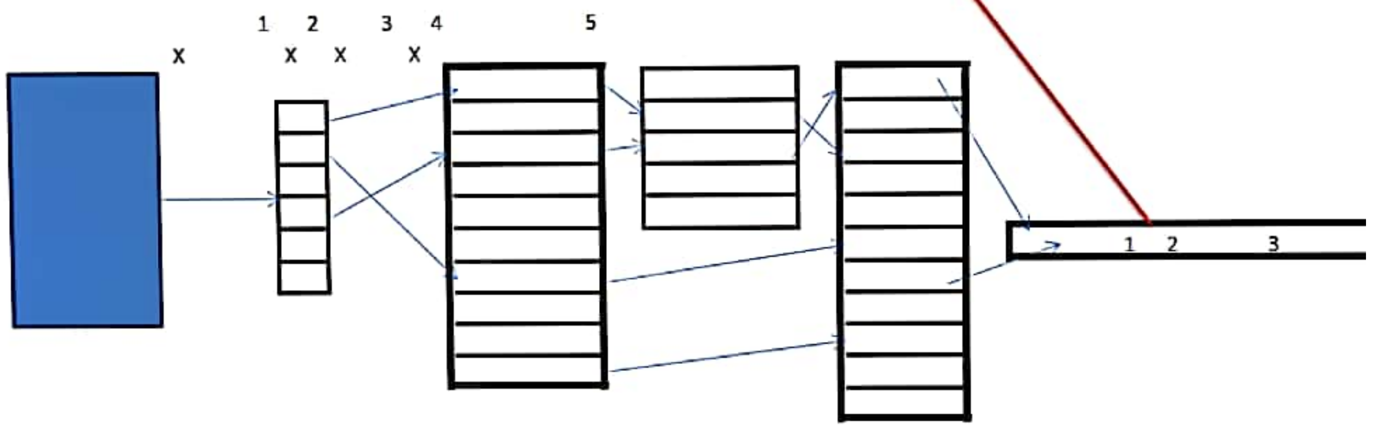
Dependency

Out of Order Execution unit

Data Hazard



Scalar CPI<1



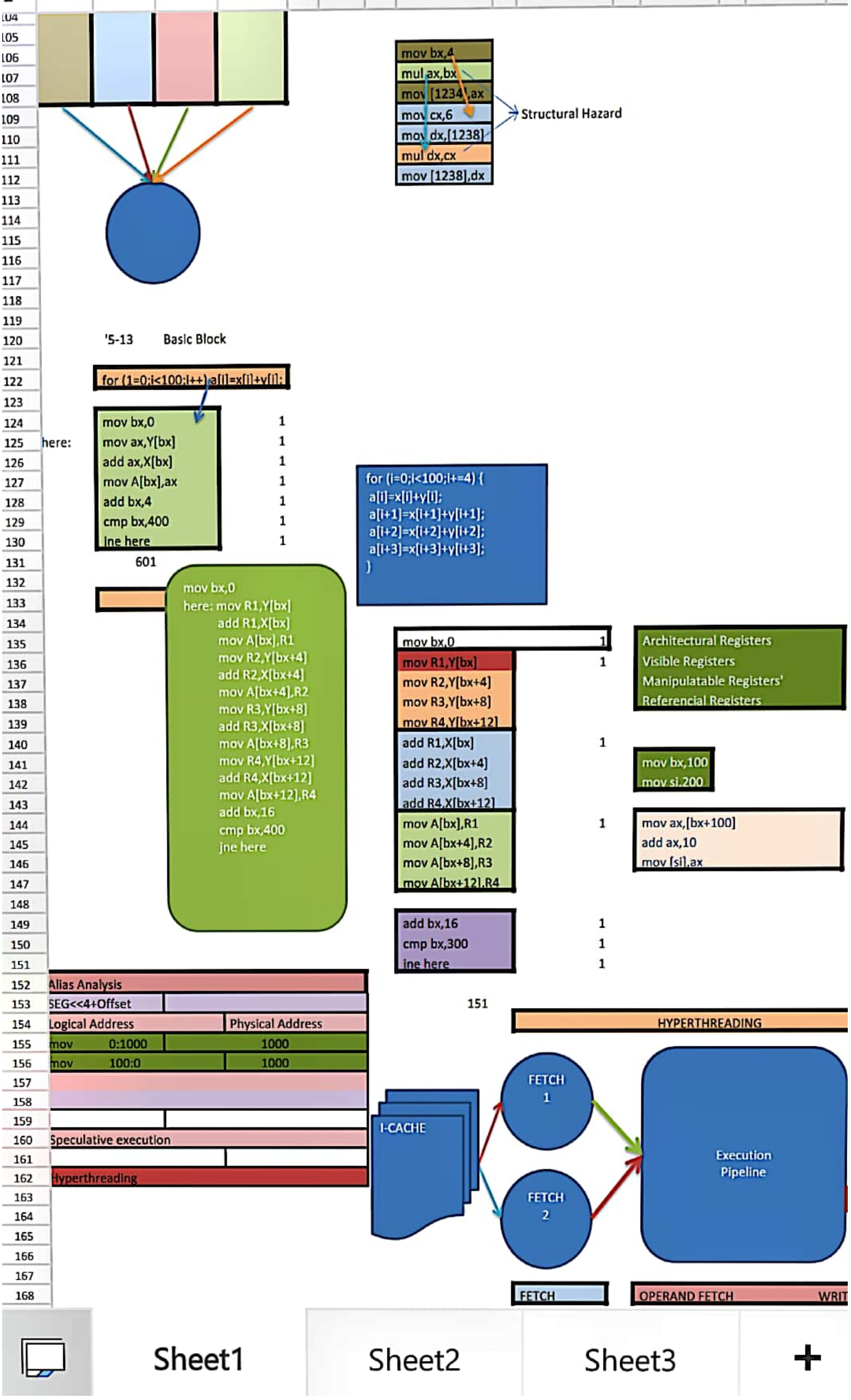
Core I7 32-256

VLIW
Very Long Instruction Word

nop	nop	nop
ins6	NOP	NOP
ins3	ins4	ins5

ins1
ins2
ins3
ins4
ins5
ins6

mov bx,4
mul ax,bx



```
mov bx,4
mul ax,bx
mov [1234],ax
mov cx,6
mov dx,[1238]
mul dx,cx
mov [1238],dx
```

Structural Hazard

'5-13   Basic Block

```
for (1=0;i<100;i++) a[i]=x[i]+y[i];
```

```
       mov bx,0              1
here:  mov ax,Y[bx]          1
       add ax,X[bx]          1
       mov A[bx],ax          1
       add bx,4              1
       cmp bx,400            1
       jne here              1
                 601
```

```
for (i=0;i<100;i+=4) {
 a[i]=x[i]+y[i];
 a[i+1]=x[i+1]+y[i+1];
 a[i+2]=x[i+2]+y[i+2];
 a[i+3]=x[i+3]+y[i+3];
}
```

```
mov bx,0
here: mov R1,Y[bx]
      add R1,X[bx]
      mov A[bx],R1
      mov R2,Y[bx+4]
      add R2,X[bx+4]
      mov A[bx+4],R2
      mov R3,Y[bx+8]
      add R3,X[bx+8]
      mov A[bx+8],R3
      mov R4,Y[bx+12]
      add R4,X[bx+12]
      mov A[bx+12],R4
      add bx,16
      cmp bx,400
      jne here
```

```
mov bx,0               1
mov R1,Y[bx]           1
mov R2,Y[bx+4]
mov R3,Y[bx+8]
mov R4,Y[bx+12]
add R1,X[bx]           1
add R2,X[bx+4]
add R3,X[bx+8]
add R4,X[bx+12]
mov A[bx],R1           1
mov A[bx+4],R2
mov A[bx+8],R3
mov A[bx+12],R4
add bx,16              1
cmp bx,300             1
jne here               1
                151
```

Architectural Registers
Visible Registers
Manipulatable Registers'
Referencial Registers

```
mov bx,100
mov si,200
```

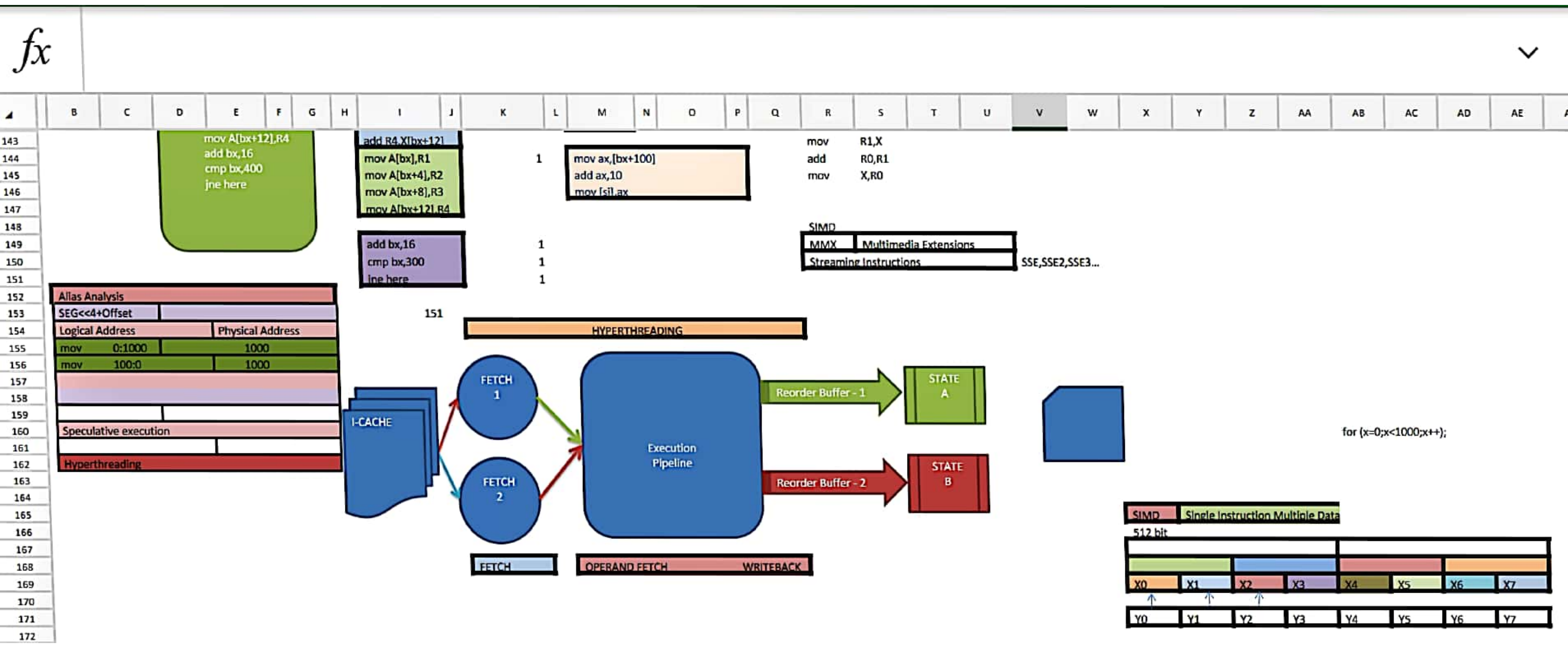
```
mov ax,[bx+100]
add ax,10
mov [si],ax
```

Alias Analysis	
SEG<<4+Offset	
Logical Address	Physical Address
mov 0:1000	1000
mov 100:0	1000
Speculative execution	
Hyperthreading	

HYPERTHREADING

FETCH

OPERAND FETCH   WRIT



```
mov A[bx+12],R4
add bx,16
cmp bx,400
jne here
```

```
add R4,X[bx+12]
mov A[bx],R1
mov A[bx+4],R2
mov A[bx+8],R3
mov A[bx+12],R4
```

```
add bx,16      1
cmp bx,300     1
jne here       1
```

151

```
1   mov ax,[bx+100]
    add ax,10
    mov [si],ax
```

```
mov   R1,X
add   R0,R1
mov   X,R0
```

SIMD

MMX	Multimedia Extensions
Streaming Instructions	

SSE,SSE2,SSE3...

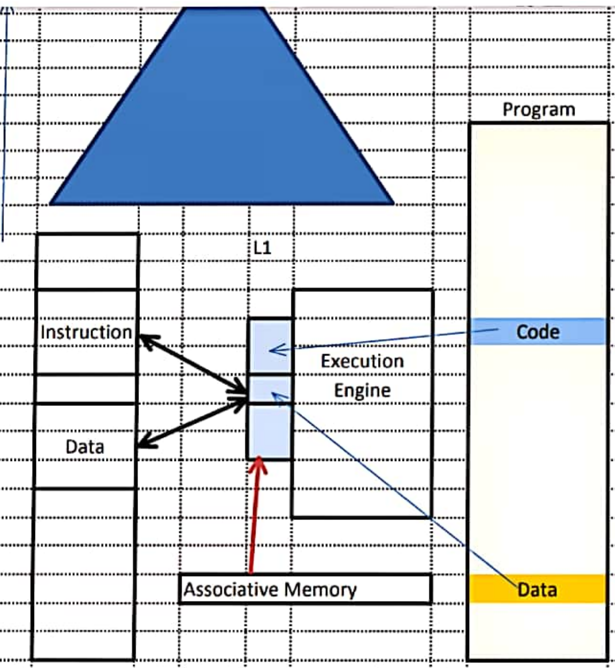
Alias Analysis	
SEG<<4+Offset	
Logical Address	Physical Address
mov 0:1000	1000
mov 100:0	1000
Speculative execution	
Hyperthreading	

for (x=0;x<1000;x++);

SIMD	Single Instruction Multiple Data

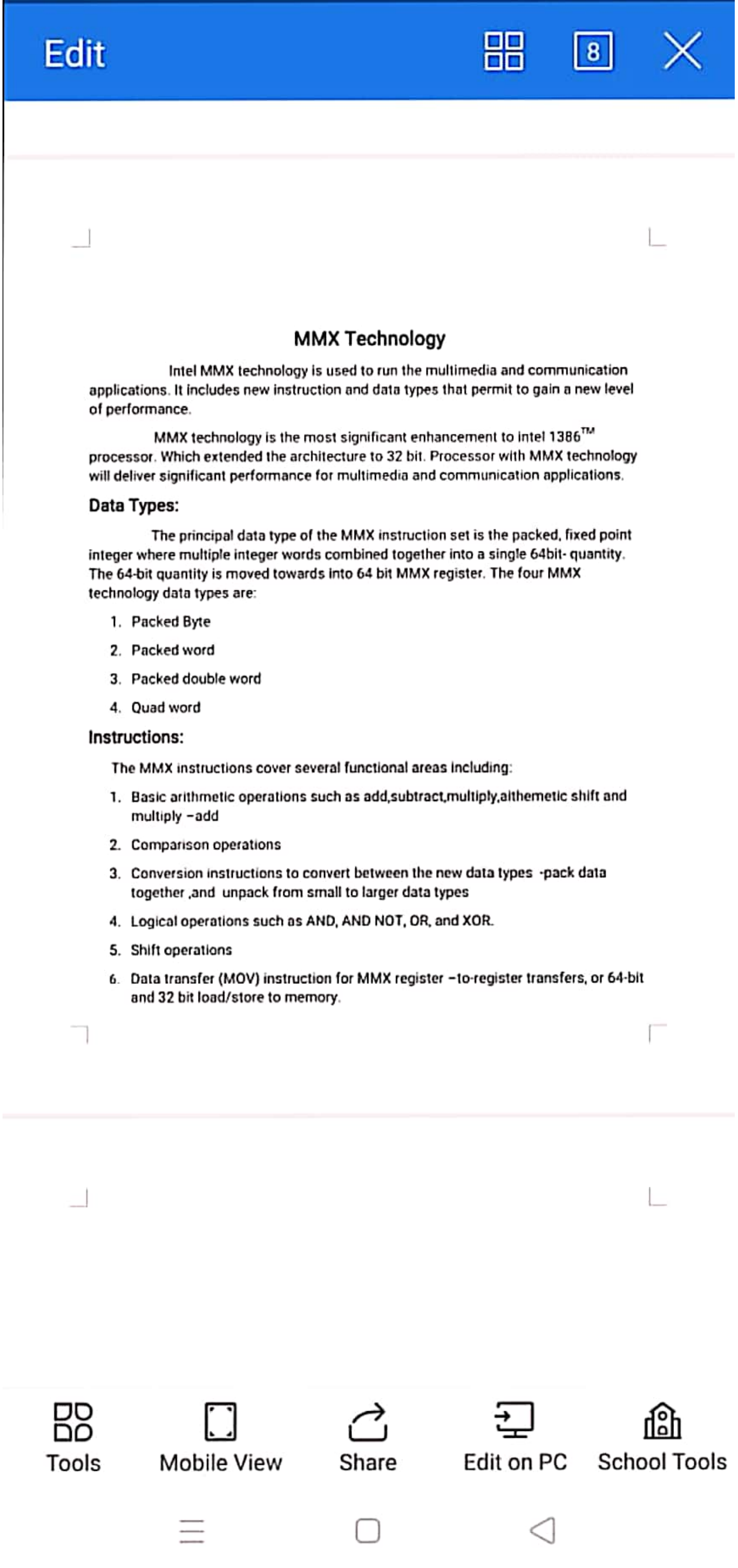
512 bit

Scanned with CamScanner



Flynn

Spatial Locality	If a program has accessed region X, it is of high probabbility that the next access of the program will be from region X
Temporal Locality	If a program has accessed region X1,X2,X3,X4, it is of high probabbility that the next accesses of the program will be from region X
Sequentiality	If a program has accessed ADDRESS X it is of high probability that the next access of the program will be from Address X+1



MMX Technology

Intel MMX technology is used to run the multimedia and communication applications. It includes new instruction and data types that permit to gain a new level of performance.

MMX technology is the most significant enhancement to intel I386™ processor. Which extended the architecture to 32 bit. Processor with MMX technology will deliver significant performance for multimedia and communication applications.

Data Types:

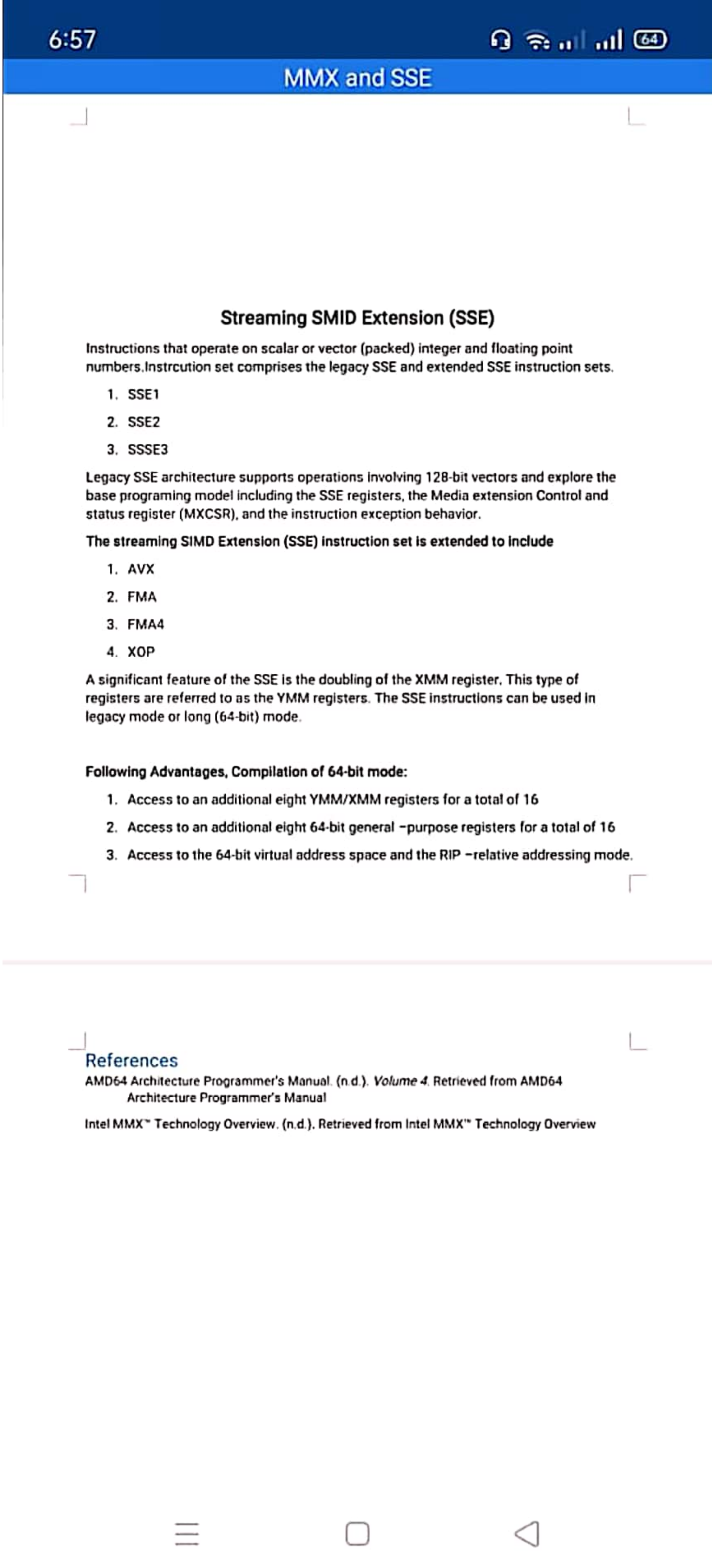
The principal data type of the MMX instruction set is the packed, fixed point integer where multiple integer words combined together into a single 64bit- quantity. The 64-bit quantity is moved towards into 64 bit MMX register. The four MMX technology data types are:

1. Packed Byte
2. Packed word
3. Packed double word
4. Quad word

Instructions:

The MMX instructions cover several functional areas including:

1. Basic arithmetic operations such as add,subtract,multiply,aithemetic shift and multiply –add
2. Comparison operations
3. Conversion instructions to convert between the new data types -pack data together ,and unpack from small to larger data types
4. Logical operations such as AND, AND NOT, OR, and XOR.
5. Shift operations
6. Data transfer (MOV) instruction for MMX register –to-register transfers, or 64-bit and 32 bit load/store to memory.



Streaming SMID Extension (SSE)

Instructions that operate on scalar or vector (packed) integer and floating point numbers.Instrcution set comprises the legacy SSE and extended SSE instruction sets.

1. SSE1
2. SSE2
3. SSSE3

Legacy SSE architecture supports operations involving 128-bit vectors and explore the base programing model including the SSE registers, the Media extension Control and status register (MXCSR), and the instruction exception behavior.

The streaming SIMD Extension (SSE) instruction set is extended to include

1. AVX
2. FMA
3. FMA4
4. XOP

A significant feature of the SSE is the doubling of the XMM register. This type of registers are referred to as the YMM registers. The SSE instructions can be used in legacy mode or long (64-bit) mode.

Following Advantages, Compilation of 64-bit mode:

1. Access to an additional eight YMM/XMM registers for a total of 16
2. Access to an additional eight 64-bit general –purpose registers for a total of 16
3. Access to the 64-bit virtual address space and the RIP –relative addressing mode.

References

AMD64 Architecture Programmer's Manual. (n.d.). *Volume 4*. Retrieved from AMD64 Architecture Programmer's Manual

Intel MMX™ Technology Overview. (n.d.). Retrieved from Intel MMX™ Technology Overview