ACA Unit-2

Conditions of Parallelism :-

ability to execute

- Several Program Seg in Parallel requires Each Segment to be independent of other segment.

Prog Seg ⇒ Can not be Executed in Parallel unless they are independent.

⇒ There are three types of dependencies of Parallelism :-

1) Data Dependencies
2) Control dependency
3) resource dependency

What is this dependency analysis :-?

Ex- let S_1 : A = B+C → Performed & result store in A
let S_2 : D = A+E → S_2 dependent on S_1.

Ex- A = B+C, D = E+F } no dependency.

(1) Data Dependency :-

5 types.

1) Flow dependency
2) Anti dependency
3) o/p dependency
4) I/o dependency
5) Unknown dependency.

(1) Flow dependency :-

Ex. $S_1 : C = a + b$

$S_2 : d = C + t$

S_2 is flow dependent on S_1

⇓

RAW hazard

Read after write hazard.

S_2 is dependent on S_1 result.

(2) Anti dependency :- reverse to flow dependency.

Ex- $S_1 : d = C + t$

$S_2 : C = a + b$

S_2 is antidependent of S_1.

means S_2 follows S_1.

⇓

WAR (Write after Read Hazard).

Ex- S_2 follows S_1 in program order and instruction S_2 tries to write a register or memory location before instruction S_1 reads, suppose S_1 is reading, and S_2 tries to overwrite. S_1 is going to read correct value but S_2 is going to write the register or memory location. So it is also called WAR Hazard.

$S_1 \not\rightarrow S_2$.

(3) O/P dependency :-

Ex- $S_1 : C = a + b$

$S_2 : C = d + e$

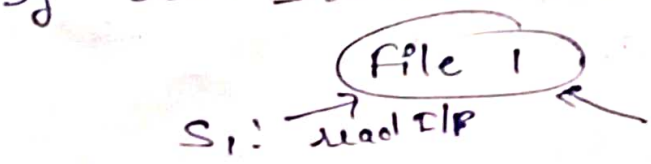
Here, S_1 and S_2 are two outputs which are storing the values in the same registers or memory location.

They are writing the same memory location so (3)
it called as WAW (write after write Hazard)

$S_1 \text{ o} \longrightarrow S_2$.

(4) I/O dependency :- Read & write are I/O statements.

I/O dependency occur not ~~between~~ because the same variable is involved. but same file is referenced by both I/O statements.



$S_1 \xrightarrow{I/O} S_2$

(5) Unknown dependence:

When dependency relation b/w two statements cannot be determined by the situation.

⇒ Parallel execution of program statement which do not have total data independence can produce non deterministic results.

46- Munendra-
-Rahul

Examples.

S_1: Load R_1, A / $R_1 \leftarrow$ Mem (A) : Load

S_2: Add R_2, R_1 / $R_2 \leftarrow (R_1) + (R_2)$

S_3: Mov R_1, R_3 / $R_1 \leftarrow (R_3)$

S_4: Store B, R_1 / Mem (B) $\leftarrow R_1$

(1) flow dependency !?

$S_1 \longrightarrow S_2$, $S_3 \rightarrow S_4$, $S_2 \rightarrow S_2$ (Result)

$R_1 + R_2 \rightarrow R_2$.

(2) Anti !- ?

S_2: Add R_2, R_1

S_3: Mov R_1 R_3

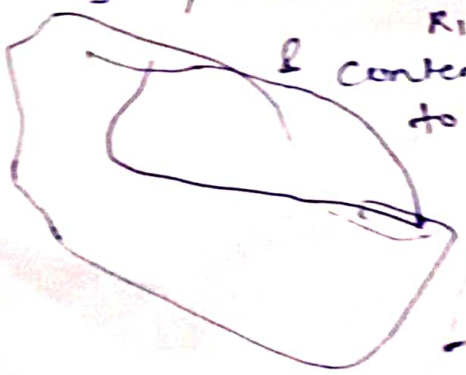
$S_2 \not\rightarrow S_3$

(3) output !-

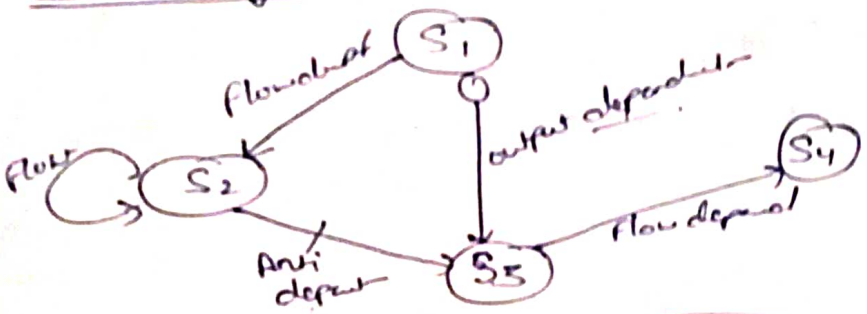
S_1: Load R_1 A

S_3: Mov R_1 R_3 — [two diff values Result is same]

$S_1 \longrightarrow S_3$, Here Content of A Loading R_1 Register & Content of R_3 Loading to R_1 Register



dependency graph:-



(2) Control dependency :-

→ This Control dependency is refers to the situation where the order of execution of statements can not be determined before runtime.

Eg: If Condition → will not be resolved until runtime, whether we check the statement the if Condition is true or not it will be resolved until runtime. where the flow of statement depends on the output on the runtime. or Conditional statements.

Control independant :-

```
Eg-  for (i=0; i<n; i++)
     {
       a[i] = c[i];
       if (a[i] < 0)
         a[i] = 1;
     }
```

Control dependent:-


```
for (i=1; i<n; i++)
{ if (a[i-1] < 0) → i-1  (i-1) < 0   0 < 0 - true.
                                Check & assign.
    a[i] = 1;
}
```

Def The control dependence always avoid Parallelism to being exploited.

Compiler techniques or all hardware branch techniques or needed to get around the control dependency.

(3) Resource Dependency :- we are trying to use two different statements or trying to use the same resources at a time. One statement depends on another statement depends on Resources.

Ex:- ⇒ data & control ⇒ based on Independence of worked to be done.

⇒ even if several statements are independent they can not be executed in parallel of they are not sufficient Processing resources.

Ex:-
```
C = a + b
d = f + k   ] they use shared memory for both because both are diff.
```

⇒ Concerned with conflicts in using shared resources registers, floating unit, ALU, memory etc.

ALU conflicts are called ALU dependance.
Memory conflicts are called storage dependency.

Hardware & Software Parallelism :-

Parallelism :
⇒ Performing more than one task at a time.
⇒ It speedup process
⇒ More work in less time.
⇒ It reduces the cost of work.

* Software ~~dependancy~~ Parallelism :- Software dependency is represented by the control and data dependency of programs. The degree of Parallelism is disclosed in the program profile or program flow graph. Software Parallelism is a function of the algorithm, programming style, and compiler optimization. Program flow graph show the pattern of simultaneously executable operation. Parallelism in a program changes during the implementation period.

* Hardware Parallelism:-

Hardware parallelism is represented by Hardware multiplicity & machine hardware. It is a function of cost & performance trade-offs. It presents the resources application design of simultaneously executable operations. It also denotes the execution of the processor resources. One method of identifying parallelism in hardware is using several instructions issued per machine cycle.

=>

* In modern comp arch implemented requires special hardware & software support for parallelism.

=>

Hardware Parallelism:- is mainly working with machine Arch. It is defined by the machine arch & hardware multiplicity.

-> It can be characterized by no of instruction that can be issued per machine cycle.

=> In Conventional processor one issue per machine cycle.

=> In modern processor 2 or more inst can be issued per machine cycle

Eg⇒ i960CA Processor was 3- issue Processor.

- arithmetic
- memory access
- 1-branch inst.

Software Parallelism:

⇒ It is defined by control & data dependency of programs.

⇒ It is a function of algorithms, Programming style, and Compiler optimization.

⇒ The Program flow graph displays the pattern of simultenously executable operation.

Difference b/w software & hardware Parallelism.

Hardware	Software
1) It is build into machine arch. and hardware multiplicity. Also known as machine parallelism.	1) Its Exploited by the Concurrent Execution of Machine language instructions in a program.
2) It is a function of cost and performance trade off.	2) Its a function of algorithm programming style and Compiler optimization.
3) It display resource utilization patterns of simultaneously executable operations. It also indicate the Peak Performance of Processor resources.	3) It displays pattern of simultaneously executable operation.

⇒ It characterized by no of instruction issue per machine cycle.

⇒ The Program flow graph displays the pattern of simultaneously executable operation.

two types:-

1) Control Parallelism:- allows 2 or more operation to be performed simultaneously

2) Data Parallelism:- almost same operation is performed over many data elements by many processes simultaneously.

Lecture - 8 (1)

Software Parallelism — Mismatch b/w S/w & H/w

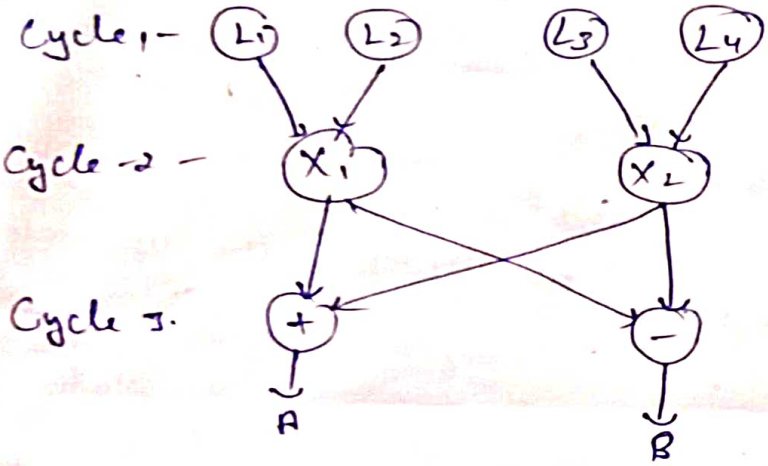
Ex-

$A = L_1 * L_2 + L_3 * L_4$

$B = L_1 * L_2 - L_3 * L_4$

4 - load instructions, & 1 - add instructions

2 - Multiply Inst, 1 - Subtract instructions



Graph of software Parallelism

L : - load operations

X_1 : Multiply operation

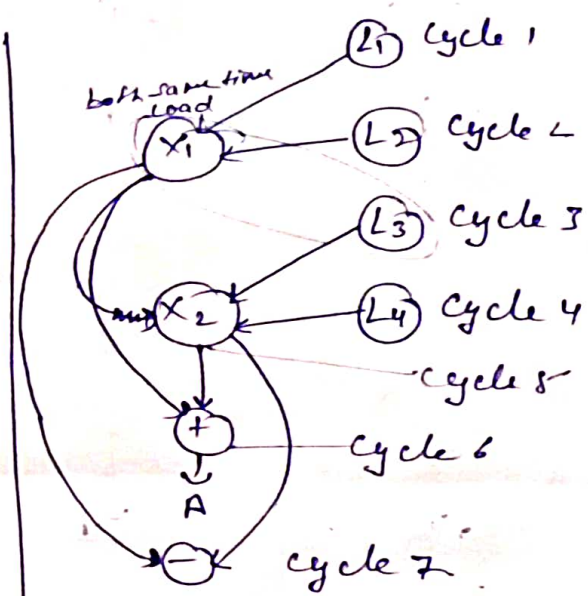
Total - 8 instructions, 3 cycles

Average s/w parallelism

$= \frac{8}{3} = 2.67$ instructions/cycle

Per machine cycle avg

More inst are execute as compare to hardware.



Avg → Total = 7

Inst = 8

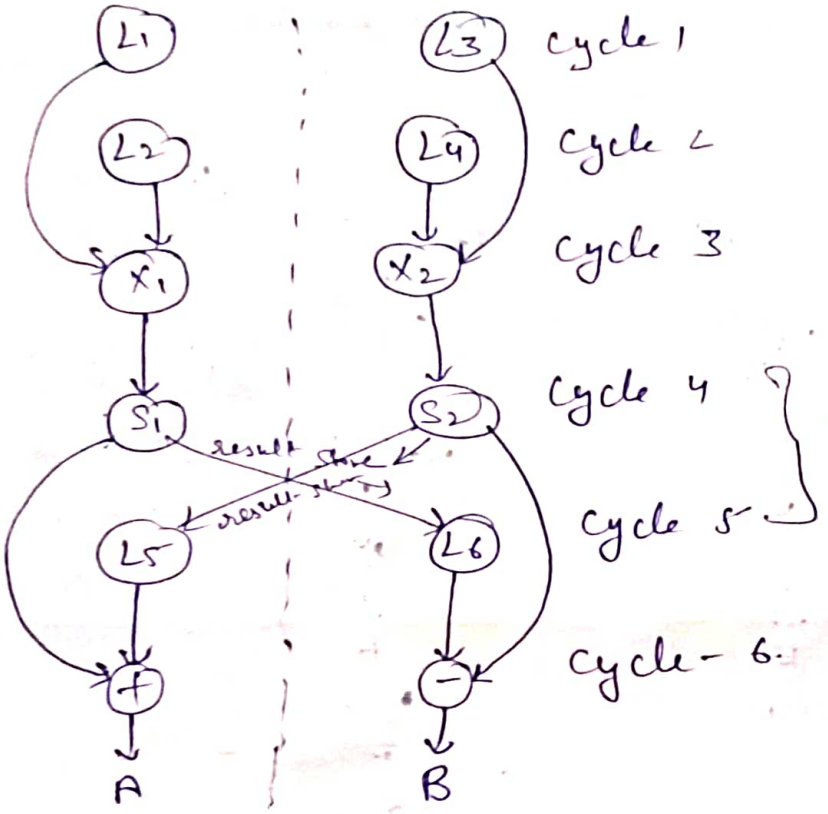
$\frac{8}{7} =$ 1.14 inst Per cycle

Here we used 2 issues Processor.

The Processor can execute one memory access (load & store) & one arithmetic operation (Multiply, add, subtract) simultaneously.

Match b/w s/w & H/w Parallelism using Dual Processor system

Dual Processor, both are Simple-issues Program



When we use two Processors no of cycles are 6. and instruction - 12.

Here, 6 Processors cycles are needed to execute 12 instruction by 2 Processors.

Avg → H/w Parallelism = $\frac{12}{6}$ = 2 instruction per cycle

→ S_1 & S_2 are two inserted store operations

→ L_5 & L_6 are two inserted load operations

These added inst are needed for interprocessor Communication through two shared Memory.

(2) Mismatch b/w s/w & H/w Parallelism:-

case (ii)

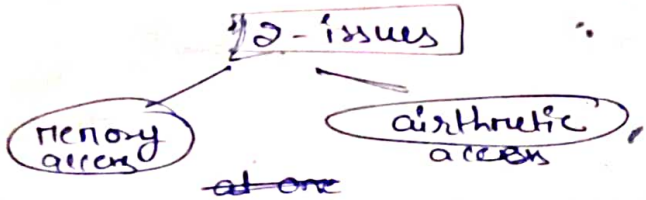
$A = L_1 * L_2 + L_3 * L_4$

$B = L_1 * L_2 - L_2 * L_4$

4 load, 4 arithmetic operations.

By a issue Processor. which can be execute only one Memory access. (load or write).

& another issues → one arithmetic operation (add, sub, mul--)



at one

* If you get to load the data, you have to get to the memory)
* and you want to store the data, you have to write on the memory.

It means → one issue perform one one access at a time.

& another issue perform only arithmetic operation.

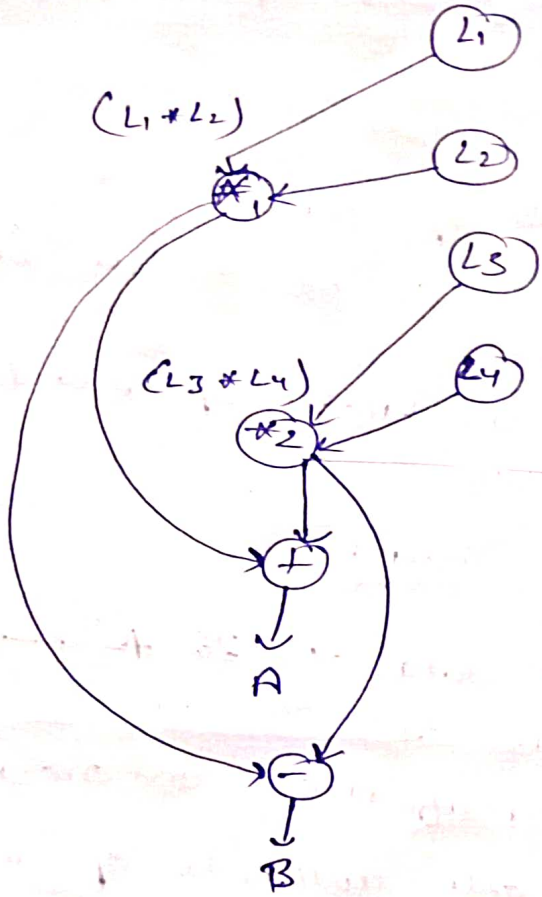
this is my hardware restrictions.]

Q→ How the Software the Programe is executed in this Hardware Parallelism? How it is going to be executed? How many avg software Parallelism get?

It is executing 7 Machine cycle

$A = L_1 * L_2 + L_3 * L_4$

$B = L_1 * L_2 - L_3 * L_4$



Cycle 1
Cycle 2
cycle
cycle 4
cycle 5
cycle 6.
cycle 7.

Total = 8 inst
cycles = 7

Avg = 8/7 = 1.14 inst Per cycle

5)