ADVANCED COMPUTER ARCHITECTURE

SESSIONAL 1

Prepared By: Neha Fotedar

Course Outline

- 8086 MAXIMUM MODE, 8087 MATHS COPROCESSOR
- 80286, 80386 AND 80486
- PARALLEL PROCESSING
- PROGRAMMING USING SHARED MEMORY
- PARALLEL ALGORITHM DESIGH AND ANALYSIS

Course Outline

Sessional 1: We will cover

- Introduction
- Different types of Parallelism
- Pipelining Hazards, BTB, BPB
- Super Pipelining
- Super Scalar Architecture
- Distributed Memory
- Shared Memory
- Symmetric Multiprocessing
- Array Processors
- Vector Processors
- Systolic Arrays

What is Parallel Computer?

- Structure of a single processor computer

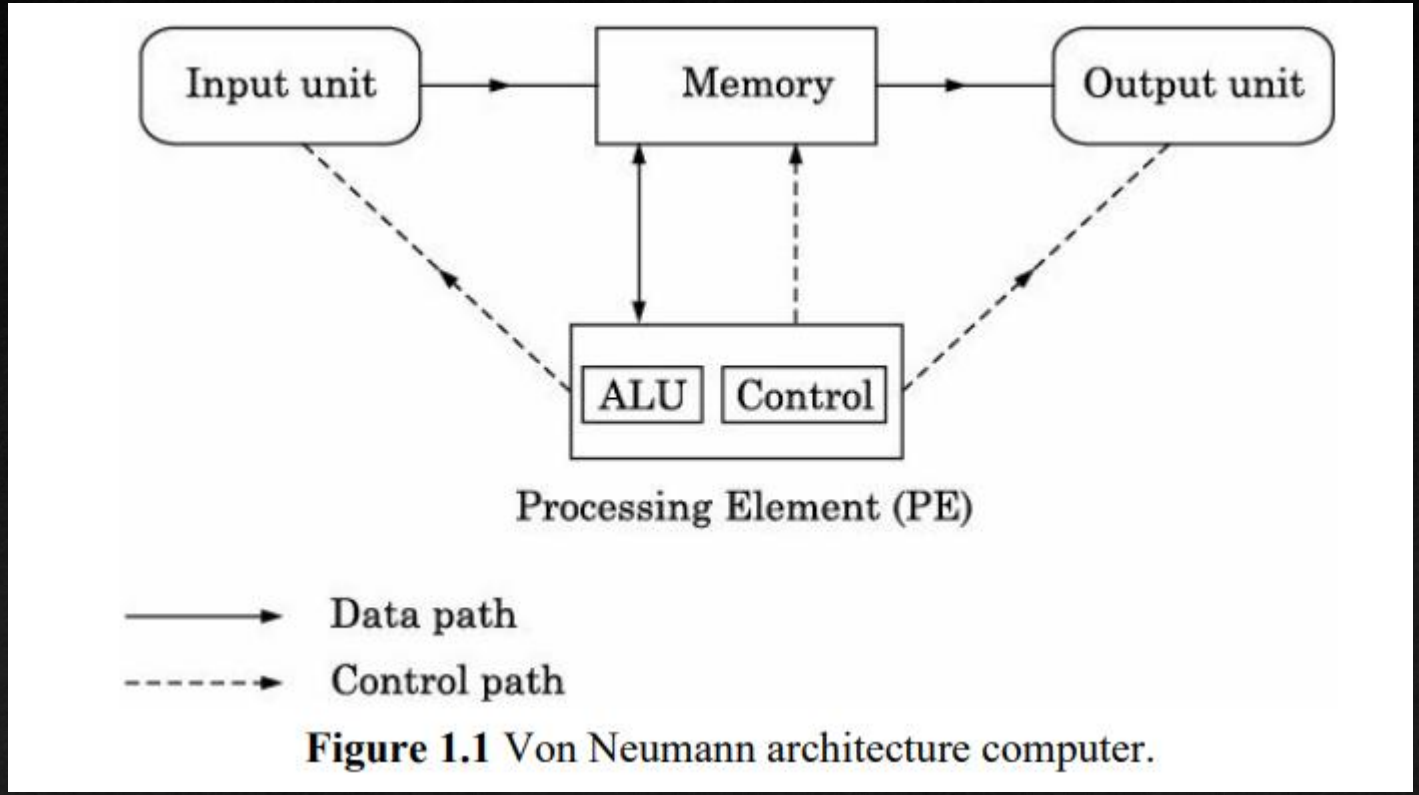


Figure 1.1 Von Neumann architecture computer.

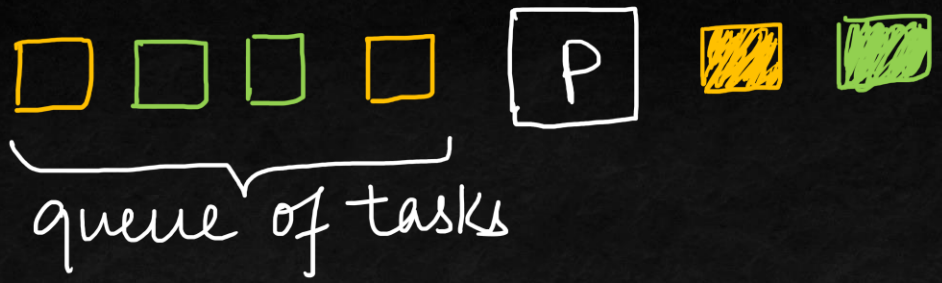
- The combination of ALU & CU is known as CPU or PE (Processing Element)

What is Parallel Computer?

- In this architecture, a program is first stored in the memory. The PE retrieves one instruction of this program at a time, interprets it and executes it. The operation of this computer is thus sequential.
- Speed Limitation?
- Speed of the PE
- How to increase the speed of data processing of the system?
- Increase the speed of the PE by increasing the clock speed.

Sequential VS Parallel

Sequential



Aim ? Increase the processing speed

Problem : we can only make this PE faster to a certain limit

Reason ? HEAT

Parallel

Sol. increase the no. of processors (PEs) & divide the workload b/w them.

Problem: perceive parallelism in algorithms & application programs should utilize parallel processing power.

What is Parallel Computer?

- A computer which consists of a number of inter-connected computers which cooperatively execute a single program to solve a problem is called a parallel computer.
- All current micro-processors are parallel processors.
- Each processor in a microprocessor chip is called a core and such a microprocessor is called a multicore processor.
- The processor retrieves a sequence of instructions from the main memory and stores them in an on-chip memory. The “cores” can then cooperate to execute these instructions in parallel.
- Even though the speed of single processor computers is continuously increasing, problems which are required to be solved nowadays are becoming more complex.

Need for High Speed Computing

- Numerical simulation to predict the behavior of physical systems.
- High performance graphics—particularly visualization, and animation.
- Big data analytics for strategic decision making.
- Synthesis of molecules for designing medicines.

Concurrency VS Parallelism

- A system is said to be concurrent if it can support two or more actions in progress at the same time.
- A system is said to be in parallel if it can support two or more actions executing simultaneously.
- Concurrency is about dealing with lots of things at once. Parallelism is about doing lot of things at once.
- Concurrency means that two or more calculations happen within the same time frame and there is usually some sort of dependency between them.
- Parallelism means that two or more calculations happen simultaneously.

Concurrency VS Parallelism

- You are asked to finish a piece of cake along with singing a song.
- You and your friend are asked to finish a piece of cake along with singing a song.
- You are given an assignment of programs in lab and told to execute the programs and note them in journal.
- Your batch is given an assignment of programs in lab and told to execute the programs and note them in journal.

Parallel VS Distributed

- Process performing computational tasks across multiple processors at once to improve computing speed and efficiency. It divides tasks into sub-tasks and executes them simultaneously through different processors.
- typically requires one computer with multiple processors.
- less scalable than distributed computing systems because the memory of a single computer can only handle so many processors at once.
- all processors share the same memory and the processors communicate with each other with the help of this shared memory.
- processors share a single master clock for synchronization.
- used to increase computer performance and for scientific computing

- Process of connecting multiple computers via a local network or wide area network so that they can act together as a single ultra-powerful computer capable of performing computations that no single computer within the network would be able to perform on its own.
- involves several autonomous (and often geographically separate and/or distant) computer systems working on divided tasks.
- always scale with additional computers.
- have their own memory and processors and communicate using message passing.
- use synchronization algorithms.
- used to share resources and improve scalability

Solving Problems in Parallel (Different types of Parallelism)

- Utilizing Temporal Parallelism
- Example: Suppose 1000 candidates appear in an examination. Assume that there are answers to 4 questions in each answer book. If a teacher is to correct these answer books, the following instructions may be given to him:
- **Procedure 2.1 Instructions given to a teacher to correct an answer book**

Step 1: Take an answer book from the pile of answer books.

Step 2: Correct the answer to *Q*1 namely, A1.

Step 3: Repeat Step 2 for answers to *Q*2, *Q*3, *Q*4, namely, *A*2, *A*3, *A*4.

Step 4: Add marks given for each answer.

Step 5: Put answer book in a pile of corrected answer books.

Step 6: Repeat Steps 1 to 5 until no more answer books are left in the input.

Solving Problems in Parallel (Different types of Parallelism)

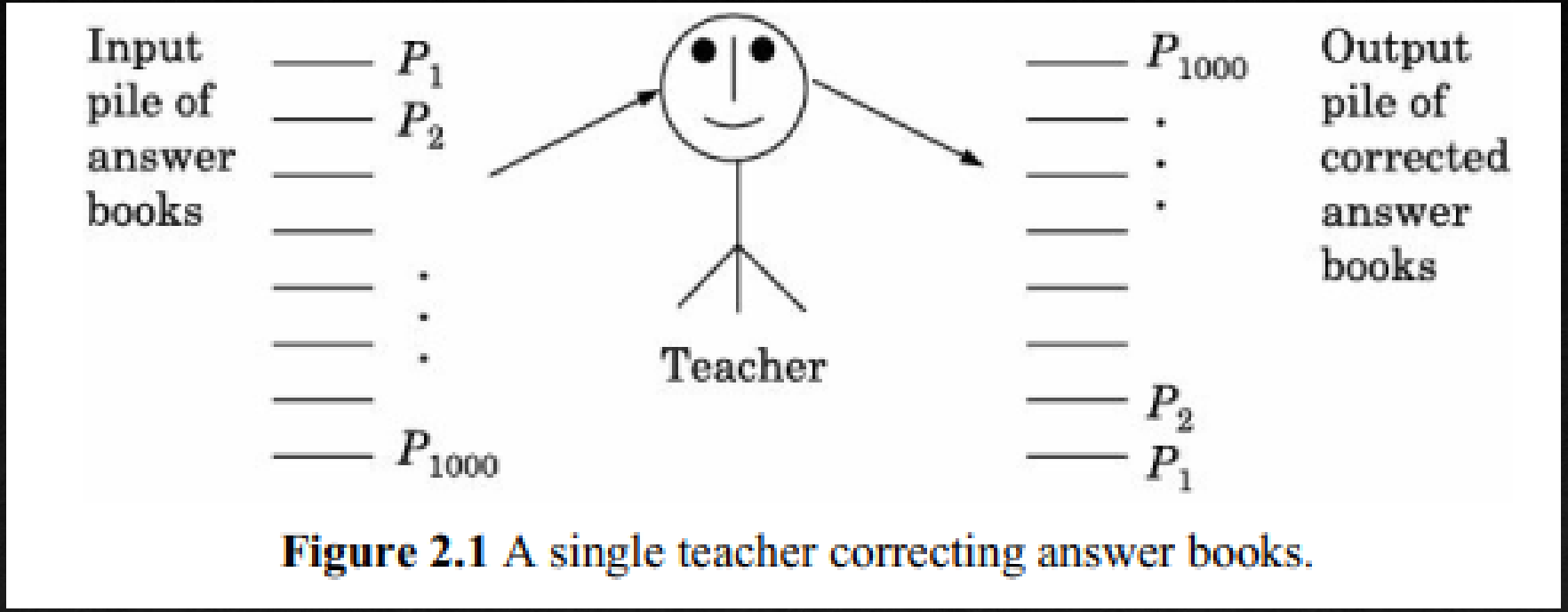


Figure 2.1 A single teacher correcting answer books.

if a paper takes 20 minutes to correct.
A single teacher would take ? 20000 minutes t correct 1000 papers.

Method 1: Temporal Parallelism

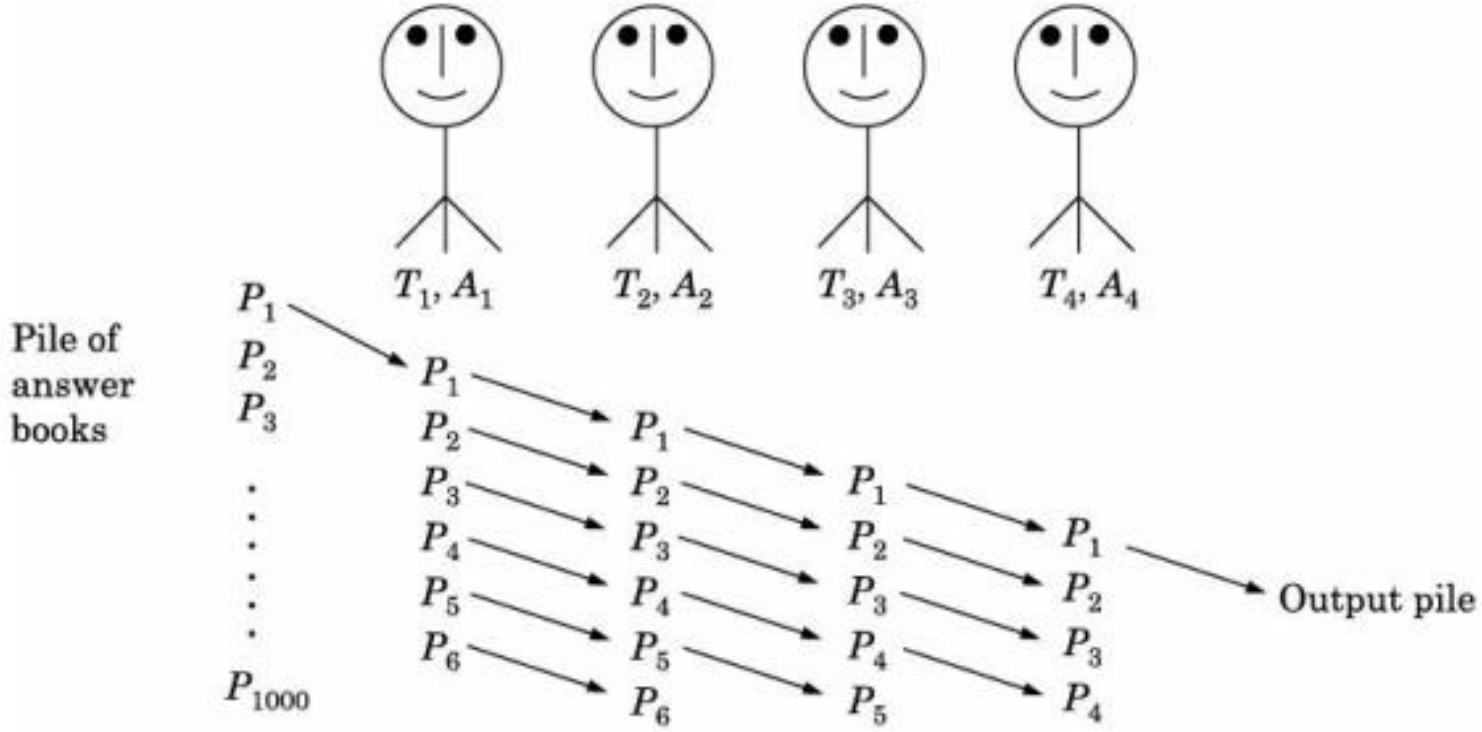


Figure 2.2 Four teachers working in a pipeline or assembly line.

Method 1: Temporal Parallelism

- This method is a *parallel processing* method as 4 teachers work in parallel, that is, simultaneously to do the job in a shorter time.
- The type of parallel processing used in this method is called *temporal parallelism.*
- The term temporal means pertaining to time.
- As this method breaks up a job into a set of tasks to be executed overlapped in time, it is said to use temporal parallelism.
- It is also known as assembly line processing, pipeline processing, or vector processing.
- The terminology vector processing is appropriate in this case if we think of an answer book as a vector with 4 components; the components are the answers *A*1, *A*2, *A*3, and *A*4.

s

Method 1: Temporal Parallelism

- This method of parallel processing is appropriate if:
  1. The jobs to be carried out are identical.
  2. A job can be divided into many *independent* tasks (i.e., each task can be done independent of other tasks) and each can be performed by a different teacher.
  3. The time taken for each task is same.
  4. The time taken to send a job from one teacher to the next is negligible compared to the time needed to do a task.
  5. The number of tasks is much smaller as compared to the total number of jobs to be done.

Method 1: Temporal Parallelism

Analysis

Let the no. of jobs = n

Let the time to do a job = P

let each job be divided into k tasks [each task is being performed by a single teacher]

Let the time for doing each task = P/k

Time to complete n jobs with no pipeline processing = nP

Time to complete n jobs with a pipeline org. of k teachers =

$$= P + (n-1)\frac{P}{k} = \frac{P(k+n-1)}{k}$$

$$\text{Speedup due to pipeline processing} = \frac{nP}{P(k+n-1)/k} = \frac{k}{1+\frac{k-1}{n}}$$

Method 1: Temporal Parallelism

$$\text{speed up} = \frac{k}{1 + \frac{k-1}{n}}$$

∴ if $n >> k$ then $(k-1)/n = 0$ and speedup is nearly $= k$.

$k = ?$ no. of tasks a job can be broken into

in our ex k is no. of teachers in pipeline

or no. of stages in pipeline.

speed up $\propto k$ (provided n is very large)

Method 1: Temporal Parallelism

- Problems encountered

1. **Synchronization**
2. **Bubbles in pipeline**
3. **Fault tolerance**
4. **Inter-task communication**
5. **Scalability**

Method 2: Data Parallelism

- In this method, we divide the answer books into four piles and give one pile to each teacher (see Fig. 2.3). Each teacher follows identical instructions given in Procedure 2.1

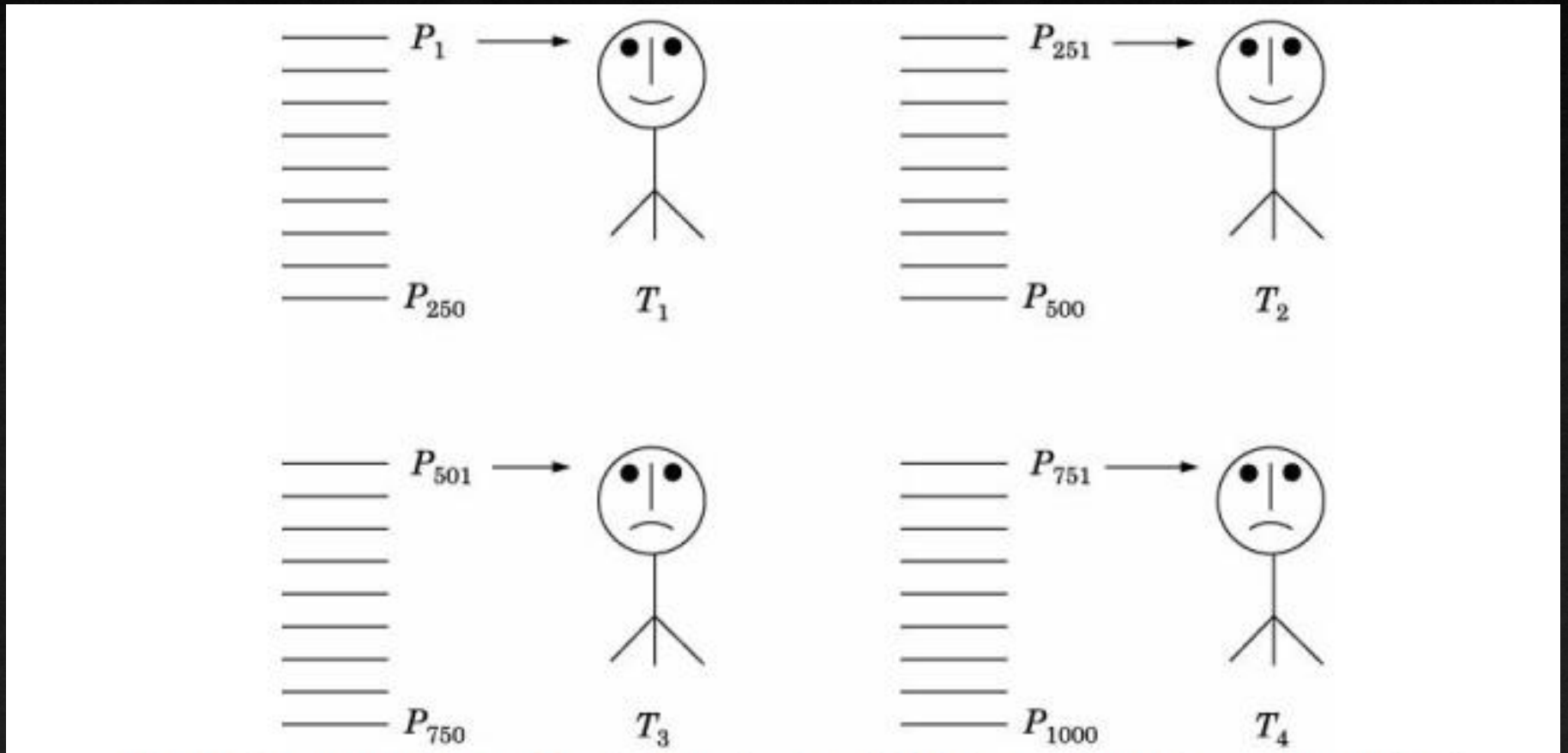


Figure 2.3 Four teachers working independently and simultaneously on 4 sets of answer books.

Method 2: Data Parallelism

Analysis: For the same example this method would take = 5000 min

Let the no. of jobs = n

Let the time to do a job = p

Let there be k tasks for one job or k teachers (eg)

Let the time to distribute the jobs to k teachers be kq.

The time to complete n jobs by a single teacher = np

The time to complete n jobs by k teachers = $kq + \frac{np}{k}$

Speed up due to parallel processing = $\frac{np}{kq + \frac{np}{k}}$

$= \frac{knp}{k^2q + np} = \frac{k}{1 + \frac{k^2q}{np}}$

Method 2: Data Parallelism

if $k^2q << np$ then the speed up is nearly equal to k
This will be true if time to distribute jobs is small.

Example: let $n = 1000$, $p = 20$ and $k = 4$. Let time to create one subset of jobs is 1.

$$\text{Speed up} = \frac{np}{kq + \frac{np}{k}} = \frac{1000 \times 20}{4 + \frac{1000 \times 20}{4}} = \frac{4 \times (1000 \times 20)}{16 + (1000 \times 20)}$$

$$= 3.997 \cong 4$$

$$\text{Efficiency} = \frac{\text{Speed up}}{k} = \frac{3.997}{4} = 0.99$$

Method 2: Data Parallelism

for the same eg. if $k = 100$ then

$$\text{speed up} = \frac{1000 \times 20}{100 + \frac{1000 \times 20}{100}} = 66.7$$

$$\text{Efficiency} = \frac{\text{Speed up}}{k} = \frac{66.7}{1000} = 0.667$$

$\Rightarrow$ speedup is not $\propto$ to k (no. of teachers)
(no. of tasks a job can be broken into)
(no. of stages in a pipeline)

Method 2: Data Parallelism

- Advantages

  1. There is *no synchronization* required between teachers. Each teacher can correct papers independently at his own pace.

  2. The problem of *bubbles* is *absent.* If a question is unanswered in a paper It only reduces the time to correct that paper.

  3. This method is *more fault tolerant.* One of the teachers can take a coffee break without affecting the work of other teachers.

  4. There is no communication required between teachers as each teacher works independently. Thus, there is no inter-task communication delay.

Method 2: Data Parallelism

- Disadvantages

  1. The assignment of jobs to each teacher is pre-decided. This is called a *static assignment*. Thus, if a teacher is slow then the completion time of the total job will be slowed down. If another teacher gets many blank answer books he will complete his work early and will thereafter be idle. Thus, a static assignment of jobs is not efficient.

  2. We must be able to divide the set of jobs into subsets of mutually independent jobs. Each subset should take the same time to complete.

  3. Each teacher must be capable of correcting answers to all questions. This is to be contrasted with pipelining in which each teacher specialized in correcting the answer to only one question.

  4. The time taken to divide a set of jobs into equal subsets of jobs should be small. Further, the number of subsets should be small as compared to the number of jobs.

Method 3: Temporal + Data

- This is called *parallel pipeline processing*. This method almost halves the time taken by a single pipeline.

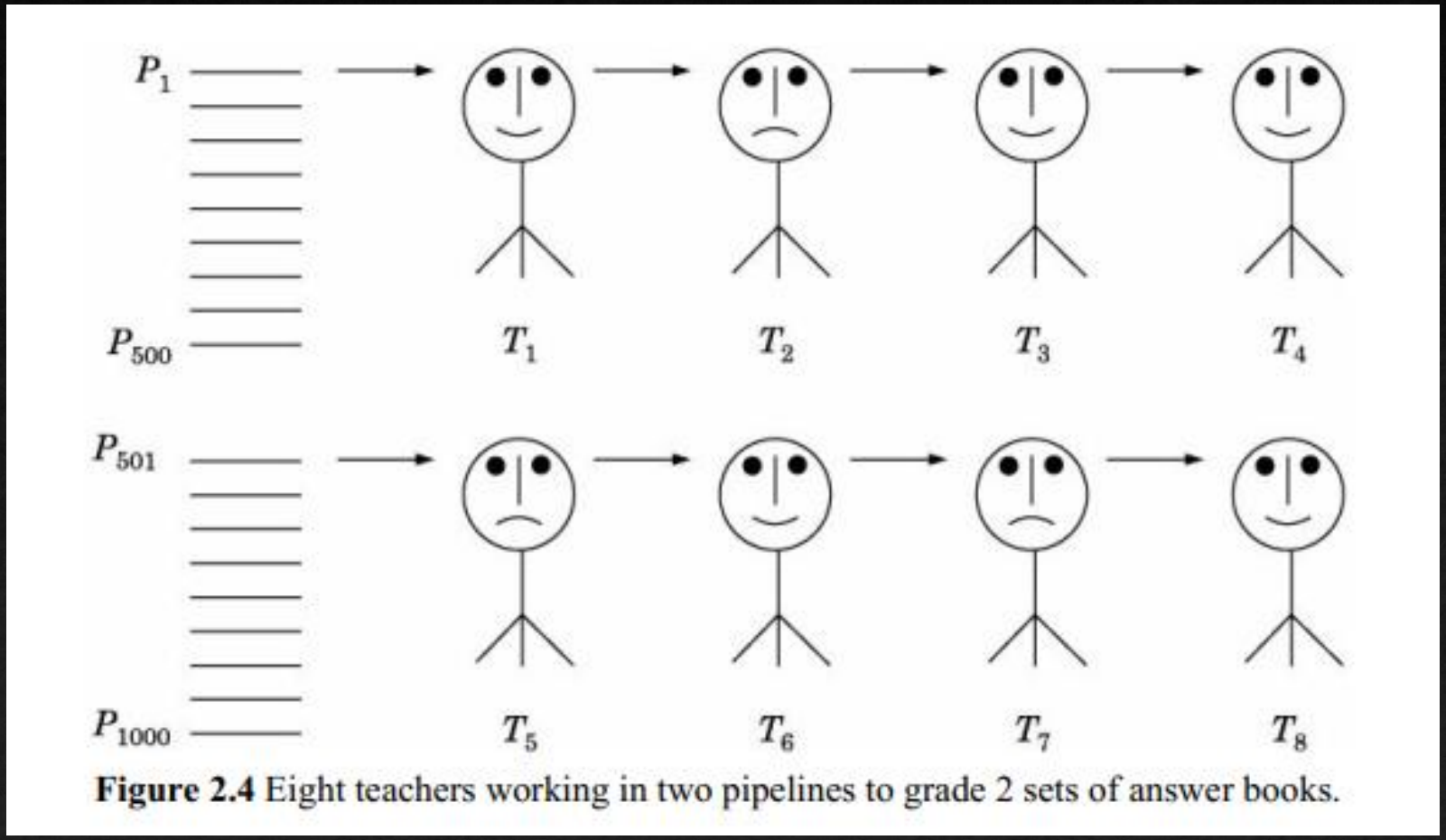


Figure 2.4 Eight teachers working in two pipelines to grade 2 sets of answer books.

Method 3: Temporal + Data

- Time taken by each pipeline?
- 2515 minutes. How?
- The method is effective only if the number of jobs given to *each* pipeline is much larger than the number of stages in the pipeline.
- Multiple pipeline processing was used in supercomputers such as Cray and NEC-SX as this method is very efficient for numerical computing in which a number of long vectors and large matrices are used as data and could be processed simultaneously.

Method 4: Data Parallelism with Dynamic Assignment

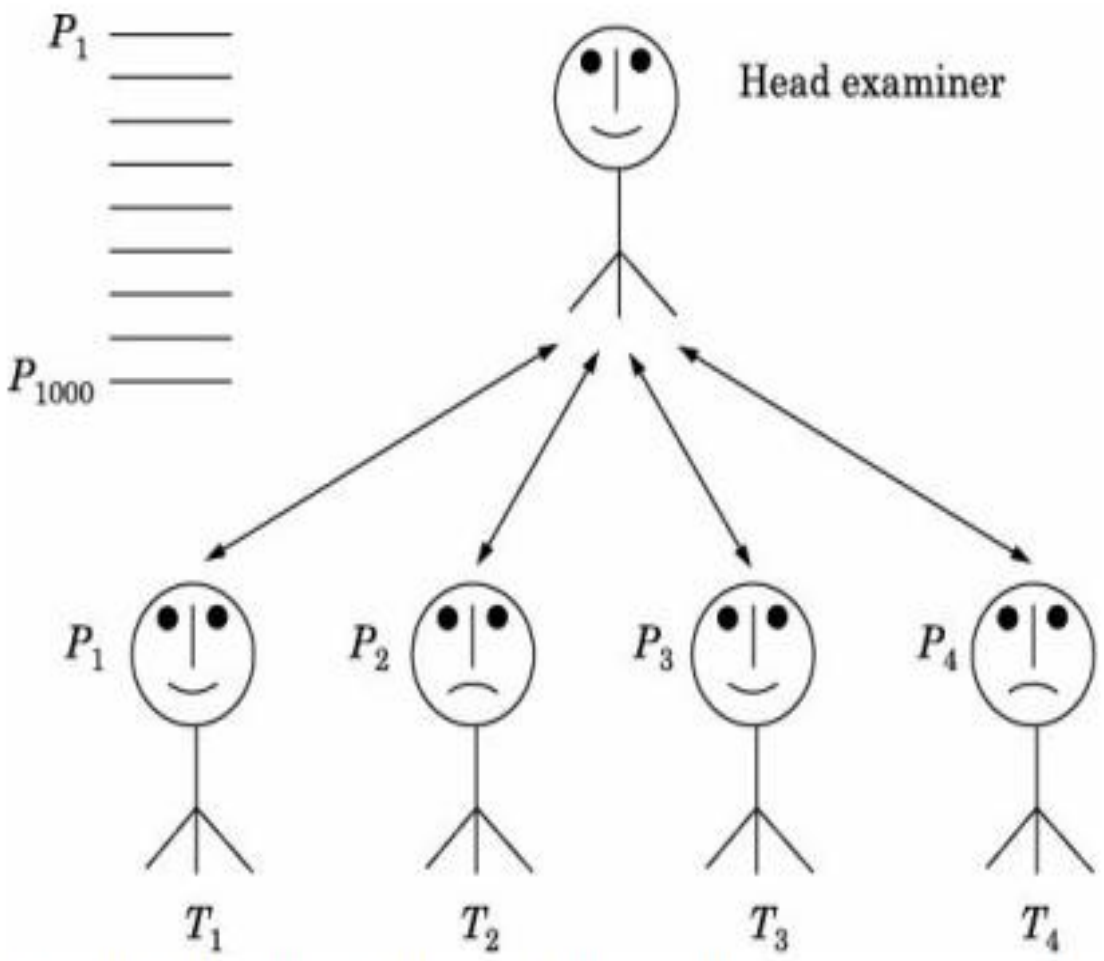


Figure 2.5 Scheduling of jobs by head examiner to a group of teachers.

- balancing of the work assigned to each teacher dynamically as work progresses.
- A teacher who finishes his work quickly gets another paper immediately and he is not forced to be idle.
- The time to correct a paper may widely vary without creating a bottleneck.
- The method is not affected by *bubbles,* namely, unanswered questions or blank answer papers.

Method 4: Data Parallelism with Dynamic Assignment

- Disadvantages

  1. If many teachers complete correcting an answer paper simultaneously only one of them will get the next paper at once as the examiner can attend to only one teacher at a time. The other teachers have to wait in a queue to get their next answer paper.

  2. The head examiner can become a bottleneck. If he takes a coffee break, all teachers will be idle! However, a teacher taking a coffee break does not cause breakdown of the system.

  3. The head examiner himself is idle between handing out papers.

  4. It is difficult to increase the number of teachers as it will increase the probability of many teachers completing their jobs simultaneously thereby leading to long queues of teachers waiting to get an answer paper.

Method 4: Data Parallelism with Dynamic Assignment

Analysis of the scalability of the dynamic assignment method.

Let the total no. of papers = n

for no parallel proc. time to correct papers = np ← time for 1 paper

let k teachers be employed to work in parallel.

Let the time a teacher waits to get an answer book (paper) from the head examiner and to return it to the head examiner = q

time taken by each teacher to get a paper, grade & return = $q+p$

Assuming that each teacher corrects (n/k) papers and all work simultaneously, the total time taken to correct all papers by k teachers = $(n/k)(q+p)$

Method 4: Data Parallelism with Dynamic Assignment

speed up due to parallel processing $= \dfrac{np}{(n/k)(q+p)}$

$= \dfrac{k}{[1 + q/p]}$

As long as $q << p$ the speed up approaches the ideal value.

If this condition is not satisfied speed up is lower.

for ex: if time to correct paper = 10 min

Time to get & return paper = 2.5 min

$$\text{speed up} = \frac{k}{1 + 2.5/10} = 0.8k$$

Efficiency loss of 20%

Method 4: Data Parallelism with Dynamic Assignment

⇒ Usually q is a function of k and goes up with k.

Why? [for our ex. a single head examiner has to cater to many teachers and he can cater to only one teacher at a time.]

Contention for a common shared Resource.

→ queue formation → increase in service time.

if $q = mk$ then speed up $= \dfrac{k}{[1 + mk/p]}$

$= \dfrac{1}{[1/k + m/p]}$

if k is very large

speed up $= p/m$

Method 4: Data Parallelism with Dynamic Assignment

speed up = P/m when k is very large

- This illustrates the point that speedup will *saturate* in such a case and will not go up as more teachers are employed.
- In other words, the time to distribute papers goes on increasing as more teachers are employed and a teacher waits longer to get a paper than to correct it! The method is thus not scalable unless (mk/p) << 1.

Method 5: Data Parallelism with Quasi-Dynamic Scheduling

- Method 4 can be improved by giving each teacher unequal sets of answer papers to correct.
- For instance, teachers 1, 2, 3, 4 may be given respectively 7, 11, 13, 19 papers initially.
- When they complete their work they may be given further small bunches of papers.
- This will randomize the job completion time of each teacher and reduce the probability of queue formation in front of the head examiner.

Method 5: Data Parallelism with Quasi-Dynamic Scheduling

- Method 5 is called *quasi-dynamic schedule* which is in between purely static and purely dynamic schedules.
- The individual jobs in a purely dynamic schedule are *fine grained* in the sense that only one job (in our example, one paper) is assigned and the time taken to complete the job is small.
- In quasi-dynamic scheduling the jobs are *coarser grained* in the sense that a bunch of answer books are given to a teacher and the time taken to complete correction will be more than if one paper is to be corrected.

Summary for Method 3 , 4 & 5

- If the loads given to processors is balanced, i.e., almost equal, then a static assignment is very good.
- One should resort to dynamic assignment of tasks to processors only if there is a wide variation in the completion time of tasks.
- The task distribution overhead in quasi-dynamic assignment with coarse grained tasks is usually much smaller than in fine grained dynamic assignment and is thus a better method.

Temporal VS Data (Parallelism)

Temporal Parallel Processing (Pipelining Idea)	Data Parallel Processing
Job is divided into a set of independent tasks and tasks are assigned for processing.	Full jobs are assigned for processing
Tasks should take equal time. Pipeline stages should thus be synchronized.	Jobs may take different times. No need to synchronize beginning of jobs.
Bubbles in jobs lead to idling of processors.	Bubbles do not cause idling of processors.
Processors specialized to do specific tasks efficiently.	Processors should be general purpose and may not do all tasks efficiently.
Task assignment static.	Job assignment may be static, dynamic, or quasi-dynamic.
Not tolerant to processor faults.	Tolerates processor faults.
Efficient with fine grained tasks.	Efficient with coarse grained tasks and quasi-dynamic scheduling.

Temporal VS Data (Parallelism)

Temporal Parallel Processing (Pipelining Idea)	Data Parallel Processing
Scales well as long as number of data items to be processed is much larger than the number of processors in the pipeline and time taken to communicate task from one processor to the next is negligible.	Scales well as long as number of jobs is much greater than the number of processors and processing time is much higher than the time to distribute data to processors.

Method 6: Specialist Data Parallelism

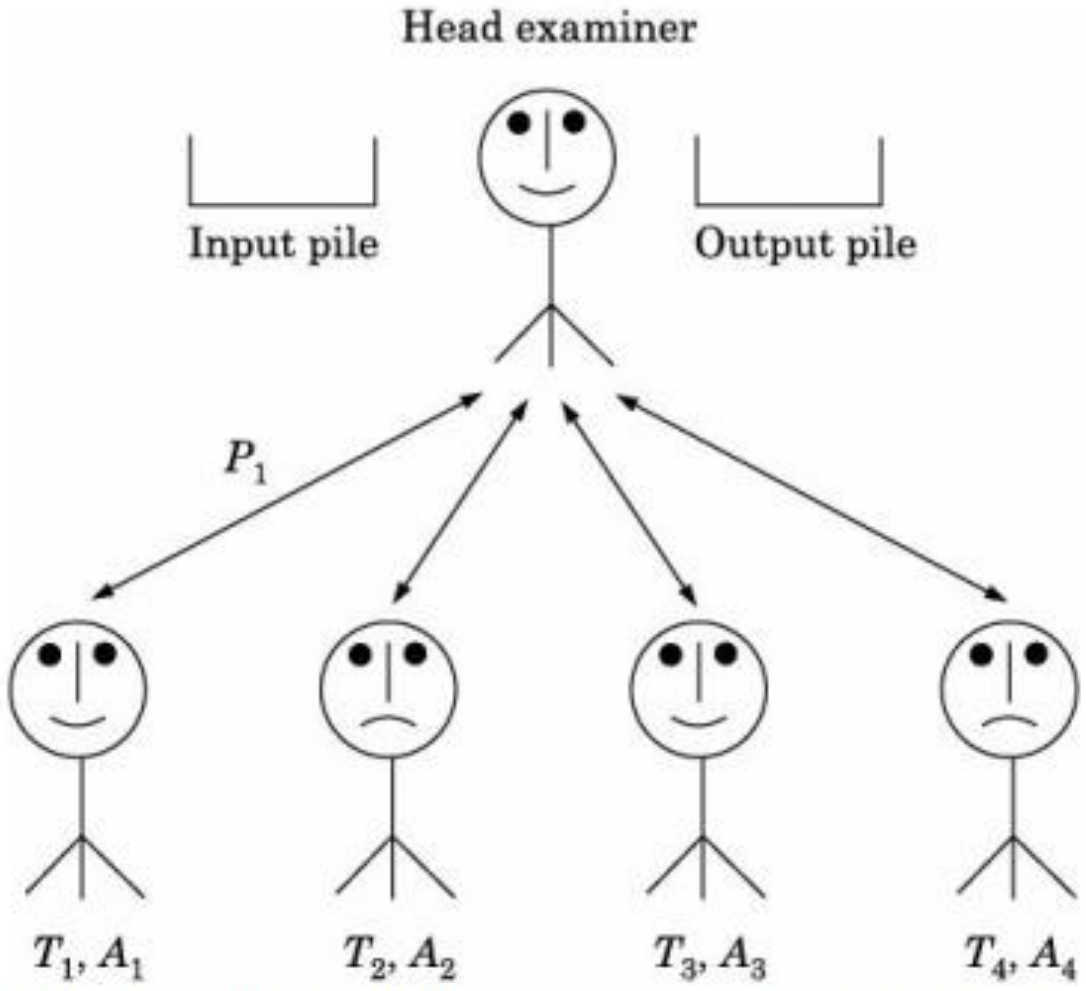


Figure 2.6 Head examiner assigning specific answers to each teacher to grade.

Method 6: Specialist Data Parallelism

- **Procedure 2.2 Task assignment method followed by the head examiner**

  1. Give one answer paper to *T*1, *T*2, *T*3, *T*4 (Remark: Teacher T_i corrects only the answer to question Q_i).

  2. When a corrected answer paper is returned check if all questions are graded. If yes add marks and put the paper in the output pile.

  3. If no, check which questions are not graded.

  4. For each *i,* if A_i is ungraded and teacher T_i is idle send it to teacher T_i *or if any other teacher* T_p is idle and an answer paper remains in input pile with A_p uncorrected send it to him.

  5. Repeat Steps 2, 3 and 4 until no answer paper remains in the input pile and all teachers are idle.

Method 6: Specialist Data Parallelism

- Problem ?
- The main problem with this method is that the load is not balanced.
- If some answers take much longer time to grade than others then some of the teachers will be busy while others are idle.
- The same problem will occur if a particular question is not answered by many students.
- Further, the head examiner will waste a lot of time seeing which questions are unanswered and which teachers are idle before he is able to assign work to a teacher.
- In other words, the maximum possible speedup will not be attained.

Method 7: Coarse Grained Specialist Temporal Parallel Processing

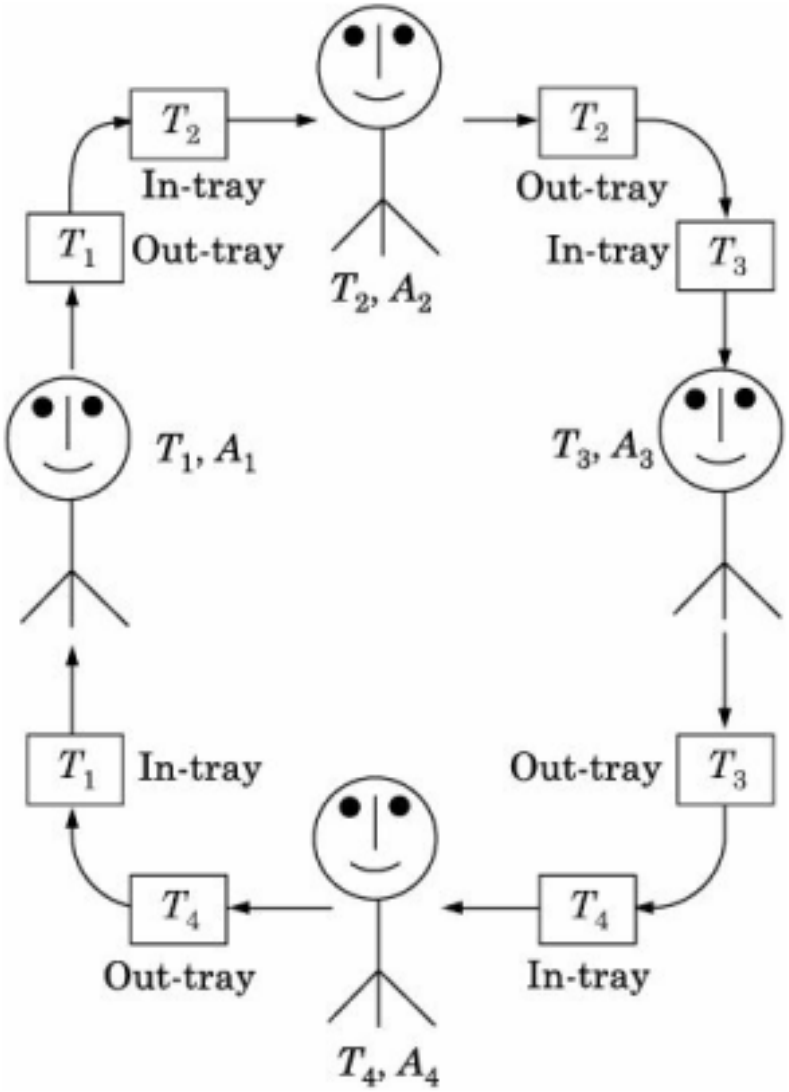


Figure 2.7 One teacher grades one answer in all papers—A circular pipeline method.

Method 7: Coarse Grained Specialist Temporal Parallel Processing

Procedure 2.3 Coarse grained specialist temporal processing

- Answer papers are divided into 4 equal piles and put in the in-trays of each teacher. Each teacher repeats 4 times Steps 1 to 5. All teachers work simultaneously.
- For teachers T_i (i = 1 to 4) do in parallel Steps 1 to 5.

  Step 1: Take an answer paper from in-tray.

  Step 2: Grade answer A_i to question Q_i and put it in out-tray.

  Step 3: Repeat Steps 1 and 2 till no papers left in in-tray.

  Step 4: Check if teacher $T_{(i + 1) \bmod 4}$'s in-tray is empty.

  Step 5: As soon as it is empty, empty own out-tray into the in-tray of that teacher.

  Step 6: All answers will be graded (in other words the procedure will terminate) when each teacher's output tray is filled 4 times.

Method 7: Coarse Grained Specialist Temporal Parallel Processing

- This section mainly illustrates the point that if processing of special tasks by special processors are to be done then balancing the load is essential and the promised speedup of parallel processing is not attainable unless this condition is fulfilled.
- It may also be observed that the method uses the concept of pipelined processing using a circular pipeline.
- Further, each stage in the pipeline has a chunk of work to do. This method does not require strict synchronization. It also tolerates bubbles due to unanswered questions in a paper or blank answer papers.

Method 8: Agenda Parallelism

- In this method an answer book is thought of as an *agenda* of answers to be graded.
- All teachers are asked to work on the first item on the agenda, namely grade the answer to the first question in all papers.
- A head examiner gives one paper to each teacher and asks him to grade the answer *A*1 to *Q*1.
- When a teacher finishes this he is given another paper in which he again grades *A*1.
- When *A*1 of all papers are graded then *A*2 is taken up by all teachers.
- This is repeated till all the answers in all papers are graded. This is data parallel method with dynamic schedule and fine grain tasks.
- This method for the problem being considered is not a good idea as the grain size is small and waiting time of teachers to receive a paper to grade may exceed the time to correct it!
- Method 4 is much better.

Summary of Methods of Parallelism

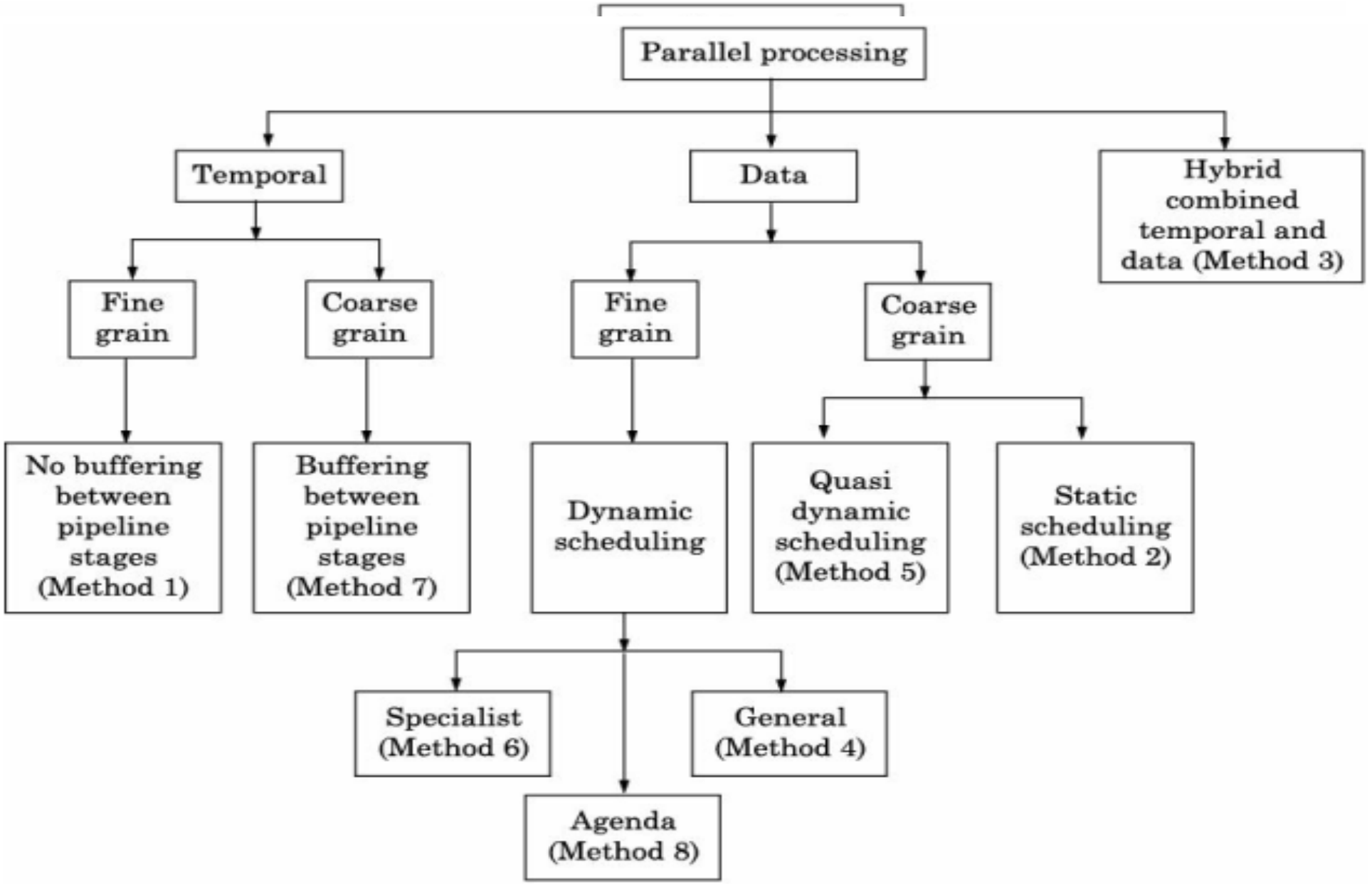
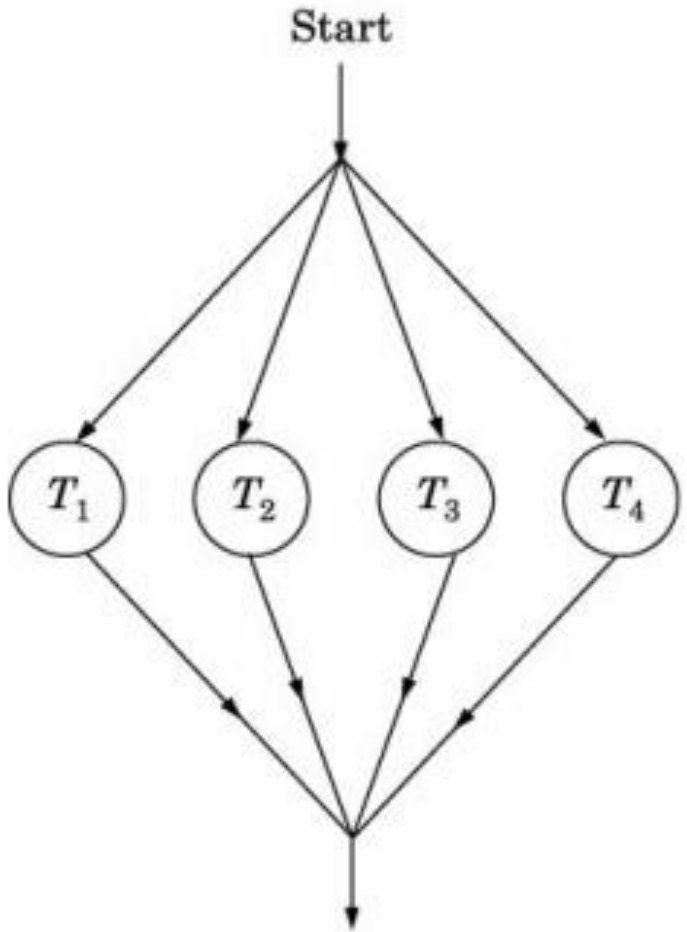


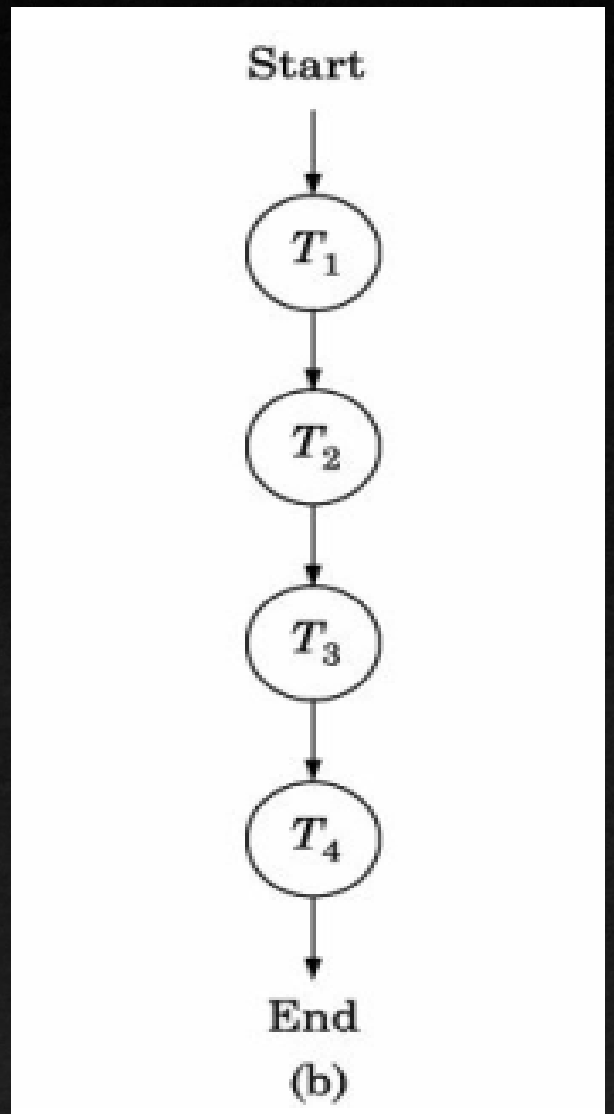
Figure 2.8 Chart showing various methods of parallel processing.

Inter-Task Dependency

- In general a job consists of tasks which are inter-related.
- Some tasks can be done simultaneously and independently while others have to wait for the completion of previous tasks.
- For example, in the grading example we have discussed, the answer to a question may depend on the answers to previous questions.
- The inter-relations of various tasks of a job may be represented graphically as a *task graph.*
- In this graph circles represent tasks which in the example we have discussed are answers to be graded.
- A line with an arrow connecting two circles shows dependency of tasks.
- The direction of an arrow shows precedence.
- A task at the head of an arrow can be done after all tasks at their respective tails are done.

Inter-Task Dependency





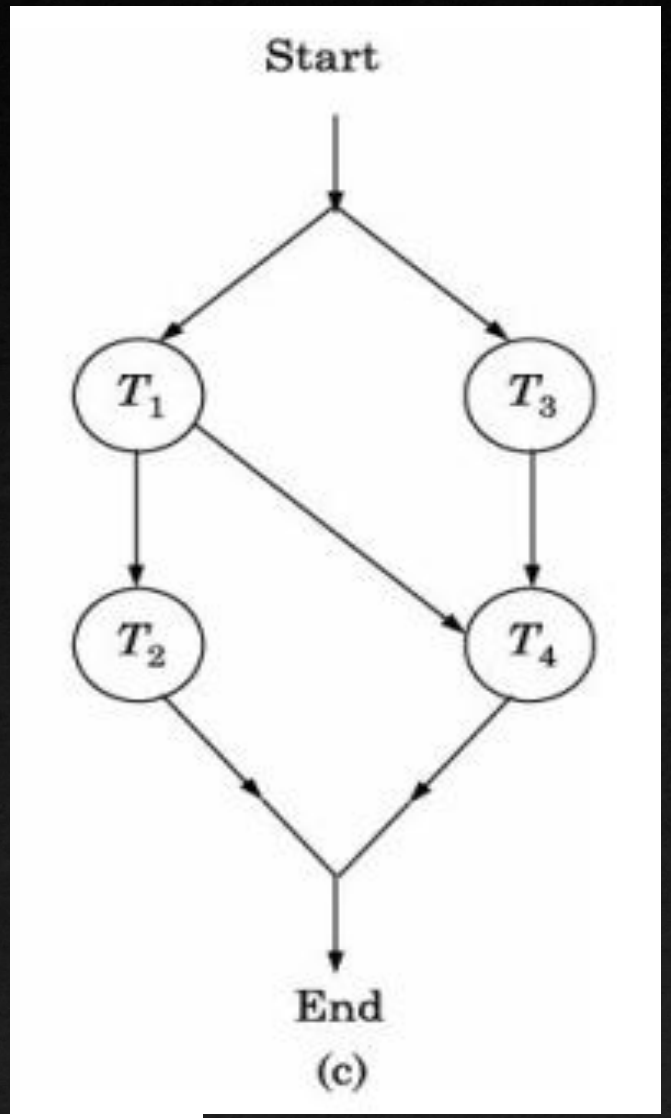


Figure 2.9 Task graphs for grading answer papers.

Inter-Task Dependency

Procedure 2.4 Recipe for Chinese vegetable fried rice

T1: Clean and wash rice

T2: Boil water in a vessel with 1 teaspoon salt

T3: Put rice in boiling water with some oil and cook till soft (do not overcook)

T4: Drain rice and cool

T 5: Wash and scrape carrots

T6: Wash and string French beans

T 7: Boil water with 1/2 teaspoon salt in two vessels

T8: Drop carrots and French beans separately in boiling water and keep for 1 minute

T9: Drain and cool carrots and French beans

T10: Dice carrots

T11: Dice French beans

T12: Peel onions and dice into small pieces.

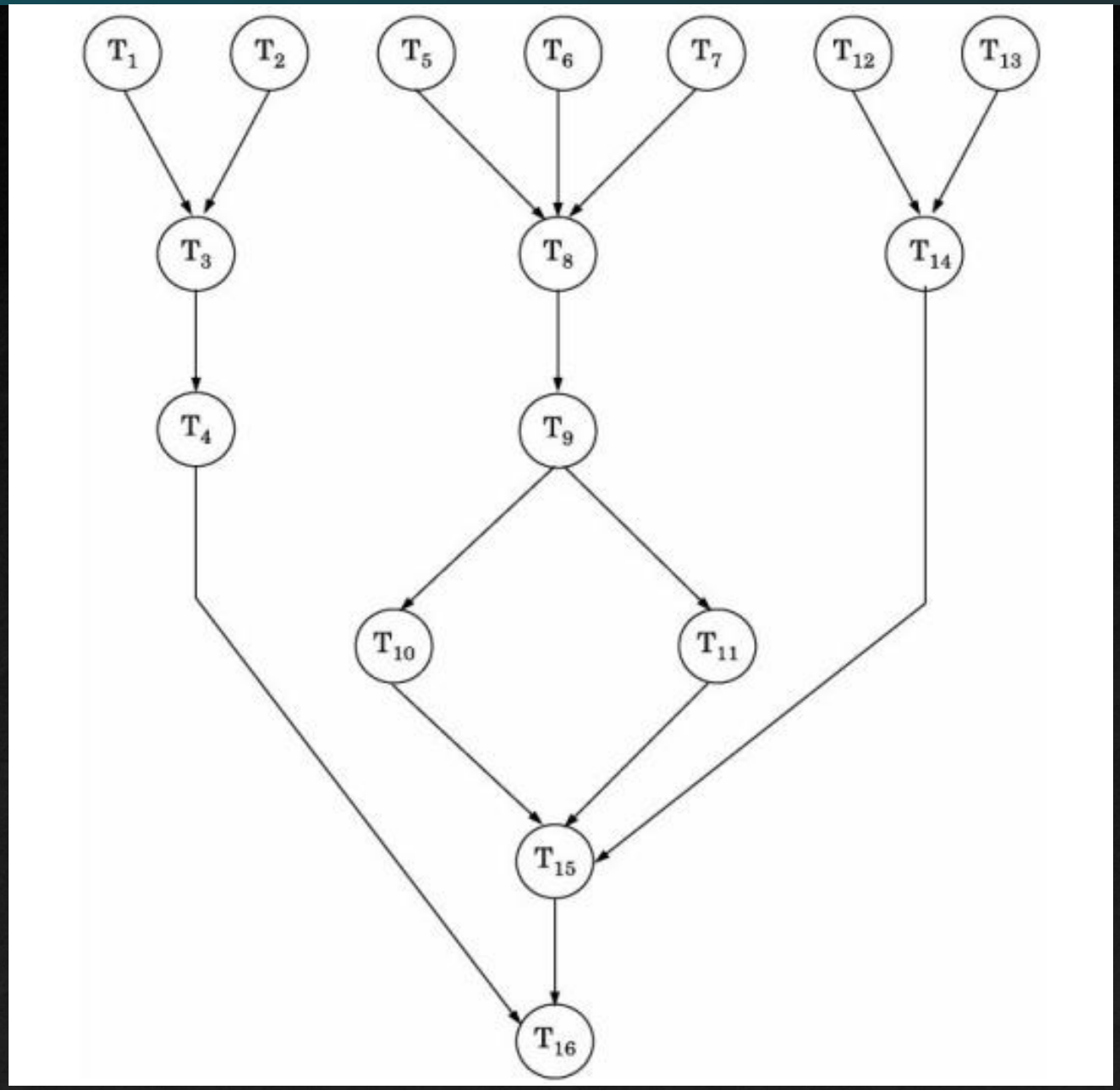
T13: Clean cauliflower. Cut into small pieces

T14: Heat oil in iron pan and fry diced onion and cauliflower for 1 minute in heated oil

T15: Add diced carrots, French beans to above and fry for 2 minutes

T16: Add cooled cooked rice, chopped spring onions, and soya sauce to the above and stir and fry for 5 minutes.

Inter-Task Dependency



Inter-Task Dependency

- Suppose this dish has to be made for 50 people and 4 cooks cooperate and cook. Tasks assigned to the cooks must be such that they work simultaneously and synchronize. The time taken for each task is given in Table 2.2.
- With these timings an assignment of tasks for each cook is arrived by using Procedure 2.5 and is shown in Fig. 2.11
- Observe that this schedule is obtained keeping the constraint of sequencing.
- This forces some cooks to be idle for sometime. In this schedule we have tried to minimize completion time of the job, but it is not the best schedule.
  We have also not attempted to equalize the load on cooks.

Inter-Task Dependency

- The minimum possible completion time requires T2, T3, T4 and T16 to be done in sequence taking 40 units of time.
- The schedule of Fig. 2.11 takes 45 units.
- If no sequencing constraints were there and if all 4 cooks work simultaneously then the minimum time is: ?
- 33 minutes

TABLE 2.2 Time for Each Task in Procedure 2.4

Task	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
Time	5	10	15	10	5	10	8	1
Task	T_9	T_{10}	T_{11}	T_{12}	T_{13}	T_{14}	T_{15}	T_{16}
Time	10	10	10	15	12	4	2	5

Inter-Task Dependency

Procedure 2.5 Assigning tasks in a task graph to cooks

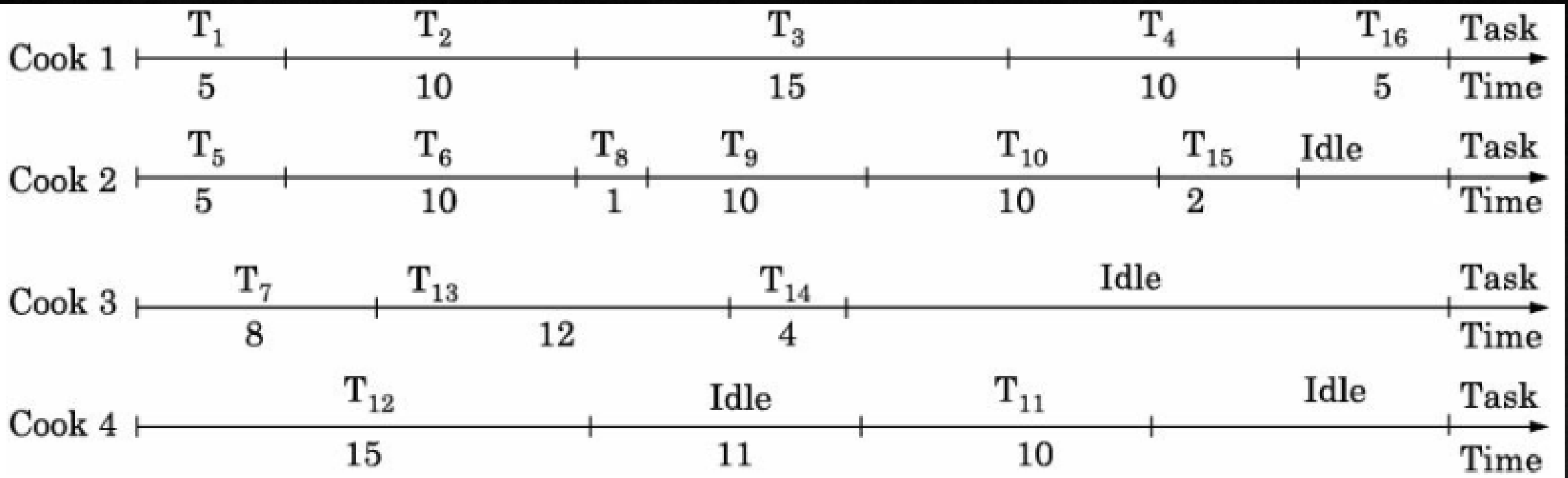


Figure 2.11 Assignment of tasks to cooks.

Instruction Level Pipelining

- A non-pipelined computer uses a 10 ns clock. The average number of clock cycles per instruction required by this machine is 4.35. When the machine is pipelined it requires a 11 ns clock. We now find out the speedup due to pipelining.

Sol. Speed up $= \dfrac{\text{non-pipelined time}}{\text{pipelined time}}$

$= \dfrac{4.35 \times 10\ ns}{11\ ns}$

$= 3.95$

DELAYS IN PIPELINE EXECUTION

- Delays in pipeline execution of instructions due to non-ideal conditions are called *pipeline hazards*.
- Available resources in a processor are limited. Structural Hazard
- Successive instructions are not independent of one another. The result generated by an instruction may be required by the next instruction. Data hazard
- All programs have branches and loops. Execution of a program is thus not in a "straight line". An ideal pipeline assumes a continuous flow of tasks. Control Hazard

Delay Due to Resource Constraints

- Pipelined execution may be delayed due to the non-availability of resources when required during execution of an instruction.
- Referring to Fig, during clock cycle 4, Step 4 of instruction i requires read/write in data memory and instruction (i + 3) requires an instruction to be fetched from the instruction memory.

Instructions	1	2	3	4	5	6	7	8	9	10
i	FI	DE	EX	MEM	SR					
i + 1		FI	DE	EX	MEM	SR				
i + 2			FI	DE	EX	MEM	SR			
i + 3				FI	DE	EX	MEM	SR		
i + 4					Fl	DE	EX	MEM	SR	
i + 5						FI	DE	EX	MEM	SR

Delay Due to Resource Constraints

- If one common memory is used for both data and instructions (as is done in many computers) only one of these instructions can be carried out and the other has to wait.
- Forced waiting of an instruction in pipeline processing is called *pipeline stall.*
- Pipeline execution may also be delayed if one of the steps in the execution of an instruction takes longer than one clock cycle. Normally a floating point division takes longer than, say, an integer addition.

Delay Due to Data Dependency

- Pipeline execution is delayed due to the fact that successive instructions are not always independent of one another. The result produced by an instruction may be needed by succeeding instructions and the results may not be ready when needed.
- Consider the following sequence of instructions

ADD R1, R2, R3	C(R3) ← C(R1) + C(R2)
MUL R3, R4, R5	C(R5) ← C(R3) * C(R4)
SUB R7, R2, R6	C(R6) ← C(R7) – C(R2)
INC R3	C(R3) ← C(R3) + 1

Fig: 3.12

9 clock cycles

clock cycle →	1	2	3	4	5	6	7	8	9	10	11
ADD R1, R2, R3	IF	DE	EX	MEM	SR						
MUL R3, R4, R5		IF	DE	X	X	EX	MEM	SR			
SUB R7, R2, R6			IF	DE	EX	MEM	SR				
INC R3				IF	DE	X	EX	MEM	SR		

Delay Due to Data Dependency

- Observe that the third instruction SUB R7, R2, R6 completes execution before the previous instruction.
- This is called *out-of-order completion* and may be unacceptable in some situations.
- Thus, many machine designers lock the pipeline when it is stalled (delayed).

Locking Pipeline due to data dependency

clock cycle →	1	2	3	4	5	6	7	8	9	10	11
ADD R1, R2, R3	IF	DE	EX	MEM	SR						
MUL R3, R4, R5		IF	DE	X	X	EX	MEM	SR			
SUB R7, R2, R6			IF	X	X	DE	EX	MEM	SR		
INC R3				X	X	IF	DE	EX	MEM	SR	

Delay Due to Data Dependency

- How we can avoid pipeline delay due to data dependency ?
- There are two methods available to do this.
- One is a hardware technique and the other is a software technique.
- The hardware method is called *register forwarding*.
- Referring to Fig. 3.12, the result of ADD Rl, R2, R3 will be in the buffer register B3. Instead of waiting till SR cycle to store it in the register file, one may provide a path from B3 to ALU input and bypass the MEM and SR cycles. This technique is called register forwarding. If register forwarding is done during ADD instruction, there will be no delay at all in the pipeline. Many pipelined processors have register forwarding as a standard feature.

Delay Due to Data Dependency

- Consider another sequence of instructions.
- Software Technique : Rescheduling of instructions

LD R2, R3, Y	$C(R2) \leftarrow C(Y + C(R3))$
ADD Rl, R2, R3	$C(R3) \leftarrow C(R1) + C(R2)$
MUL R4, R3, Rl	$C(R1) \leftarrow C(R4) \times C(R3)$
SUB R7, R8, R9	$C(R9) \leftarrow C(R7) - C(R8)$

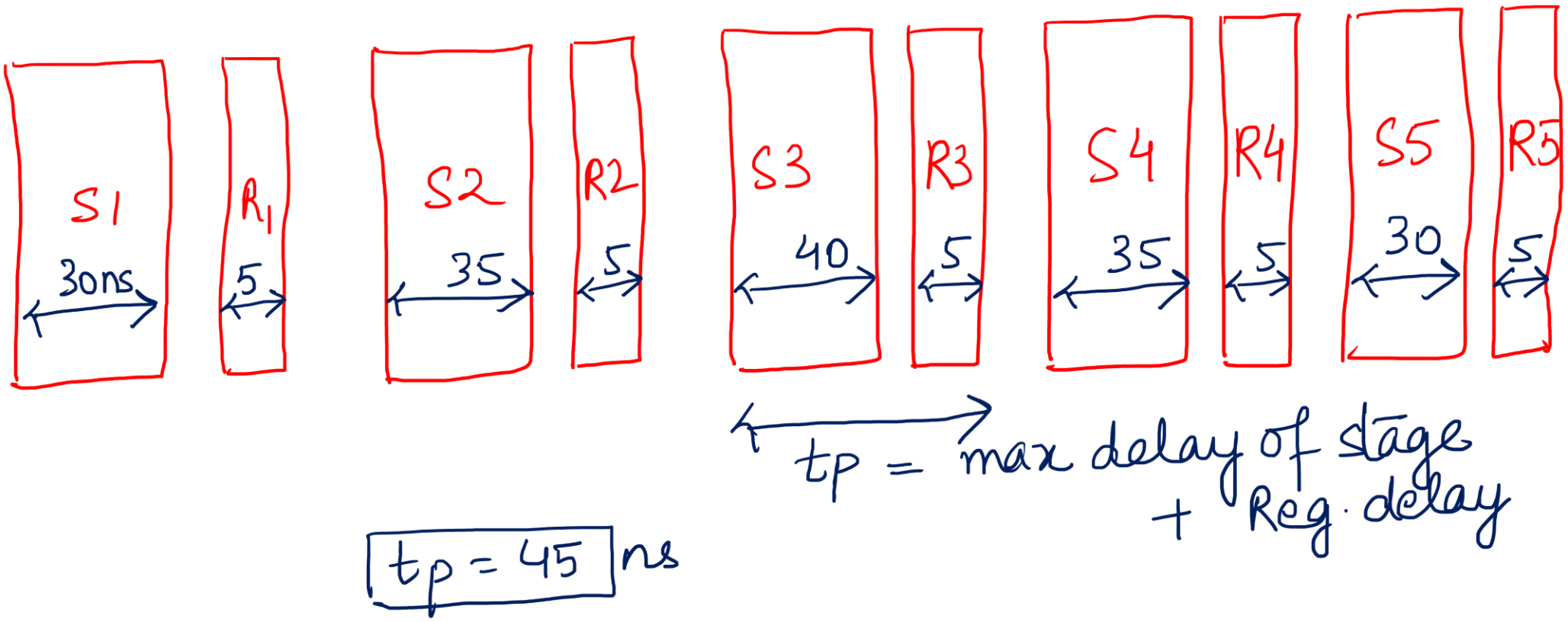
Clock cycle →	1	2	3	4	5	6	7	8	9
LD R2, R3, Y	FI	DE	EX	MEM	SR				
SUB R7, R8, R9		FI	DE	EX	MEM	SR			
ADD R1, R2, R3			FI	DE	X	EX	MEM	SR	
MUL R4, R3, R1				FI	DE	X	EX	MEM	SR

(b)

Figure 3.14 Reducing pipeline stall by software scheduling.

GATE Questions

- In a pipelined system we have 5 stages S1 = 30 ns , S2 = 35 ns, S3 = 40 ns, S4 = 35 ns and S5 = 30 ns. There is a register in between each segment with a delay of 5 ns. What is the maximum delay *i.e.*, tp?



- In a non-pipelined system, one task takes 100ns to complete. The corresponding pipelined system has 5 segment with clock pulse of 55 ns. We have to process 1000 tasks. What is the speed up factor?

NP: one task (instⁿ) time = 100 ns

$\therefore T_{NP}$ (Time taken for $n = 1000$) $= (1000 \times 100)$ ns

Pipelined (P): $m = 5$ (no. of segments or stages)

$t_p = 55$ ns (clock pulse)

$$T_p = (m + n - 1) \times t_p = (5 + 999) \times 55 \text{ ns}$$

$$\therefore \text{Speed up} = \frac{T_{NP}}{T_p} = \frac{1000 \times 100}{1004 \times 55} = \boxed{1.811 \text{ ns}}$$

- A 4-stage pipeline has the stage delays as 150, 120 , 160 and 140 ns respectively. Registers that are used between the stages have a delay of 5 ns each. Assuming constant clocking rate, the total time taken to process 1000 data items on this pipeline will be?

A. 120.4 microseconds

B. 160.5 microseconds

C. 165.5 microseconds ✓

D. 590.0 microseconds

T_p = Time taken to process 1000 data items

$= (m + n - 1) \times t_p$

$= (4 + 999) \times t_p$ (165 ns) $= 165495$ ns $= 165.5\mu s$

t_p = max delay in pipeline stages + Reg buffer delay

$= \max(150, 120, 160, 140) + 5$ ns

$= 165$ ns

- Consider a non-pipelined processor with a clock rate of 2.5 gigahertz and average cycles per instruction of four. The same processor is upgraded to a pipelined processor with five stages; but due to the internal pipelined delay, the clock speed is reduced to 2 gigahertz. Assume that there are no stalls in the pipeline. The speed up achieved in this pipelined processor is 3.2.

NP: $t_{NP} = \frac{1}{2.5\,GHz}$

T_{NP} = Total time for inst^n
= no. of clock cycles $\times$ t_{NP}
= $4 \times \frac{1}{2.5\,GHz}$

Pipelined (P): $t_p = \frac{1}{2\,GHz}$

$T_p = \frac{1}{2\,GHz}$

Speed up: $\frac{T_{NP}}{T_p} = \frac{4 \times \frac{1}{2.5\,GHz}}{\frac{1}{2\,GHz}} = \frac{8}{2.5} = 3.2$

- Consider the following processors (ns stands for nanoseconds). Assume that the pipeline registers have zero latency.

max

P1:Four-stage pipeline with stage latencies 1 ns, 2 ns, 2 ns, 1 ns. 2ns

P2:Four-stage pipeline with stage latencies 1 ns, 1.5 ns, 1.5 ns, 1.5 ns. 1.5ns

P3:Five-stage pipeline with stage latencies 0.5 ns, 1 ns, 1 ns, 0.6 ns, 1 ns. 1ns

P4: Five-stage pipeline with stage latencies 0.5 ns, 0.5 ns, 1 ns, 1 ns, 1.1 ns. 1.1ns

P5: Six-stage pipeline with stage latencies 1 ns, 2 ns, 2 ns, 2.5ns, 2.4 ns, 1.5 ns 2.5ns

Arrange the processors in ascending order of respective clock frequencies.

clock frequencies

$P_1 = \frac{1}{2ns} = 0.5\ GHz$

$P_2 = \frac{1}{1.5ns} = 0.67\ GHz$

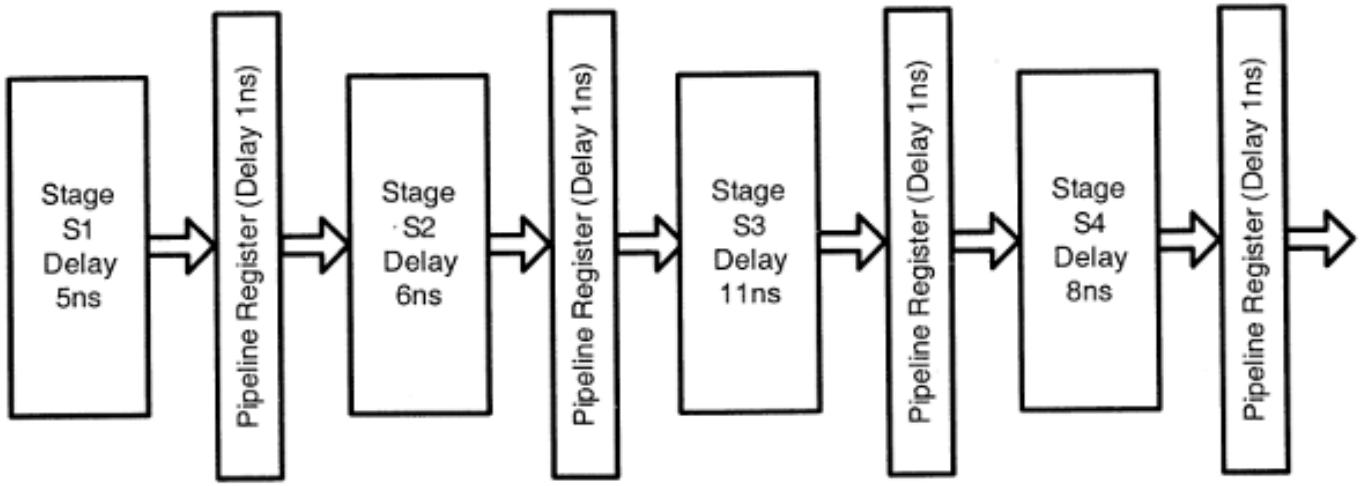
$P_3 = \frac{1}{1ns} = 1\ GHz$

$P_4 = \frac{1}{1.1ns} = 0.91\ GHz$

$P_5 = \frac{1}{2.5ns} = 0.4\ GHz$

$< P_5, P_1, P_2, P_4, P_3 >$

- Consider an instruction pipeline with four stages (S1, S2, S3 and S4) each with combinational circuit only. The pipeline registers are required between each stage and at the end of the last stage. Delays for the stages and for the pipeline registers are as given in the figure.



- What is the approximate speed up of the pipeline in steady state under ideal conditions when compared to the corresponding non-pipeline implementation?

A. 4.0
B. 2.5
C. 1.1
D. 3.0

Speed up = Time taken in non-pipelined / Time taken in pipelined

for NP time taken for each instn = (5 + 6 + 11 + 8) ns = 30 = 2.5 ns

(11 + 1) ns

we have to consider slowest stage + Reg delay

- The instruction pipeline of a RISC processor has the following stages: Instruction Fetch (IF), Instruction Decode (ID), Operand Fetch (OF), Perform Operation (PO) and Write back (WB). The IF, ID, OF and WB stages take 1 clock cycle each for every instruction. Consider a sequence of 100 instructions. In the PO stage, 40 instructions take 3 clock cycles each, 35 instructions take 2 clock cycles each, and the remaining 25 instructions take 1 clock cycle each. Assume that there are no data hazards and no control hazards. The number of clock cycles required for completion of execution of the sequence of instructions is _____.

A. 219

B. 104

C. 115

D. 220

✓ No data & control hazard
= we have to keep in mind structural hazard.

eg: if we have seq of 3 instns and I_2 takes 3 cycles for PO then

I_1	IF	ID	OF	PO	WB				
I_2		IF	ID	OF	PO	PO	PO	WB	
I_3			IF	ID	OF	–	–	PO	WB
	1	2	3	4	5	6	7	8	9

∴ if PO takes 3 cycles it induced 2 cycles of wait ie (3-1)

for ques.

Total cycles = (5 + 99) + 40×(3-1) + 35×(2-1) + 25×(1-1)

(5 + 99): 5 cycles of 1 instn; Rem instn 1 cycle

40×(3-1): 40 instns would induce delays of 2 cycles

35×(2-1): delay of 1 cycle

25×(1-1): no delay — These are normal instns

= 219 cycles

- Instruction execution in a processor is divided into 5 stage, *Instruction Fetch* (IF), *Instruction decode* (ID), *Operand Fetch* (OF), *Execute* (EX), and *Write Back* (WB). These stages take **5, 4, 20, 10**,and **3 nanoseconds (ns)** respectively. A pipelined implementation of the processor requires buffering between each pair of consecutive stages with a delay of **2 ns**. Two pipelined implementations of the processor are contemplated;
  - (i) a naive pipeline implementation (NP) with 5 stages and
  - (ii) an efficient pipeline (EP) where the OF stage is divided into stages OF1 and OF2 with execution times of **12 ns** and **8ns** respectively.
- The speedup achieved by EP over NP in executing 20 independent instructions with no hazards is

A. 1.50-1.51
B. 1.51-1.52
C. 1.52-1.53
D. 1.53-1.54

naive pipeline : $m=5$ $T_{NP} = \max(5,4,20,10,3) + 2ns$
$= 22ns$

Efficient : EP : $m=6$ $T_{EP} = \max(5,4,12,8,10,3) + 2ns$
$= 14ns$

$n=20$

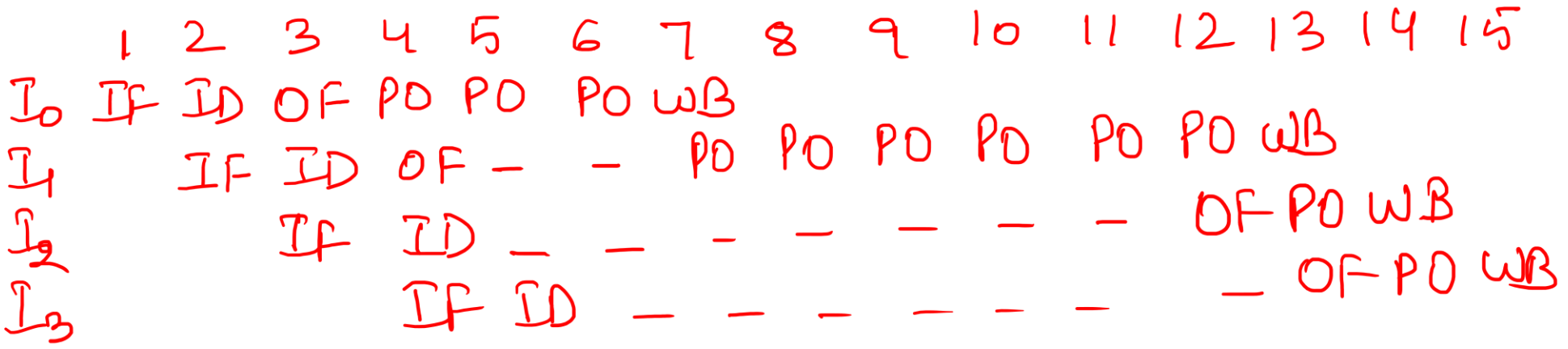
$$\text{speed up} = \frac{(5+19) \times T_{NP}}{(6+19) \times T_{EP}} = \frac{528}{350} = 1.50857$$

(T_{NP}: 22, T_{EP}: 14)

- A 5-stage pipelined processor has Instruction Fetch (IF), Instruction Decode (ID), Operand Fetch (OF), Perform Operation (PO) and Write Operand (WO) stages. The IF, ID, OF and WO stages take 1 clock cycle each for any instruction. The PO stage takes 1 clock cycle for ADD and SUB instructions, 3 clock cycles for MUL instruction, and 6 clock cycles for DIV instruction respectively. Operand forwarding is used in the pipeline. What is the number of clock cycles needed to execute the following sequence of instructions?

Instruction	Meaning of instruction
I_0 :MUL R_2 ,R_0 ,R_1	$R_2 \leftarrow R_0 * R_1$
I_1 :DIV R_5 ,R_3 ,R_4	$R_5 \leftarrow R_3 / R_4$
I_2 : ADD R_2 ,R_5 ,R_2	$R_2 \leftarrow R_5 + R_2$
I_3 :SUB R_5 ,R_2 ,R_6	$R_5 \leftarrow R_2 - R_6$

A. 13
B. 15
C. 17
D. 19



- Consider a 4 stage pipeline processor. The number of cycles needed by the four instructions I1, I2, I3, I4 in stages S1, S2, S3, S4 is shown below:

	S1	S2	S3	S4
I1	2	1	1	1
I2	1	3	2	2
I3	2	1	1	3
I4	1	2	2	2

What is the number of cycles needed to execute the following loop?

for (i=1 to 2) {I1; I2; I3; I4;}

A. 16

B. 23

C. 28

D. 30

Do it urself

- Consider a pipelined processor with the following four stages:

  IF: Instruction Fetch
  ID: Instruction Decode and Operand Fetch
  EX: Execute
  WB: Write Back

```
ADD  R2, R1,  R0      R2 ← R1 + R0
MUL  R4, R3, R2       R4 ← R3 * R2
SUB  R6, R5, R4       R6 ← R5 - R4
```

  The IF, ID and WB stages take one clock cycle each to complete the operation. The number of clock cycles for the EX stage depends on the instruction. The ADD and SUB instructions need 1 clock cycle and the MUL instruction needs 3 clock cycles in the EX stage. Operand forwarding is used in the pipelined processor. What is the number of clock cycles taken to complete the following sequence of instructions?

A. 7
B. 8
C. 10
D. 14

- A CPU has five stages pipeline and runs at 1GHz frequency. Instruction fetch happens in the first stage of the pipeline. A conditional branch instruction computes the target address and evaluates the condition in the third stage of the pipeline. The processor stops fetching new instruction following a conditional branch until the branch outcome is known. A program executes 10^9 instructions out of which 20% are conditional branches. If each instruction takes one cycle to complete on average, then total execution time of the program is

A. 1.0 second

B. 1.2 second

C. 1.4 second

D. 1.6 second

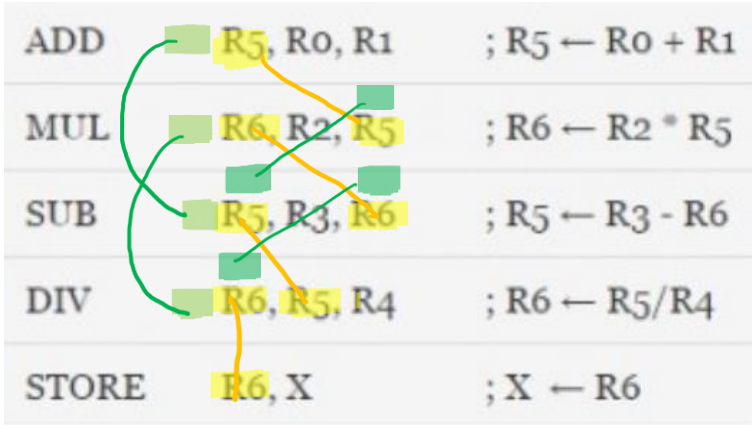
$T_C = \frac{1}{10^9} s = 10^{-9} s = 1 ns$

$10^9 + \frac{20}{100} \times 10^9 \times (3-1)$

$= 10^9 + 4 \times 10^8 = 10^8 (10+4)$

$= 1.4$ seconds

- A pipelined processor uses a 4-stage instruction pipeline with the following stages: Instruction fetch (IF), Instruction decode (ID), Execute (EX) and Writeback (WB). The arithmetic operations as well as the load and store operations are carried out in the EX stage. The sequence of instructions corresponding to the statement X = (S – R * (P + Q))/T is given below. The values of variables P, Q, R, S and T are available in the registers R_0, R_1, R_2, R_3 and R_4 respectively, before the execution of the instruction sequence. The number of Read-After-Write (RAW) dependencies, Write-After-Read(WAR) dependencies, and Write-After-Write (WAW) dependencies in the sequence of instructions are, respectively,



RAW dependency = 4

WAR dependency = 2

WAW dependency = 2

A. 2,2,4

B. 3,2,3

C. 4,2,2

D. 3,3,2

- Previous ques contd:

The IF, ID and WB stages take 1 clock cycle each. The EX stage takes 1 clock cycle each for the ADD, SUB and STORE operations, and 3 clock cycles each for MUL and DIV operations. Operand forwarding from the EX stage to the ID stage is used. The number of clock cycles required to complete the sequence of instructions is

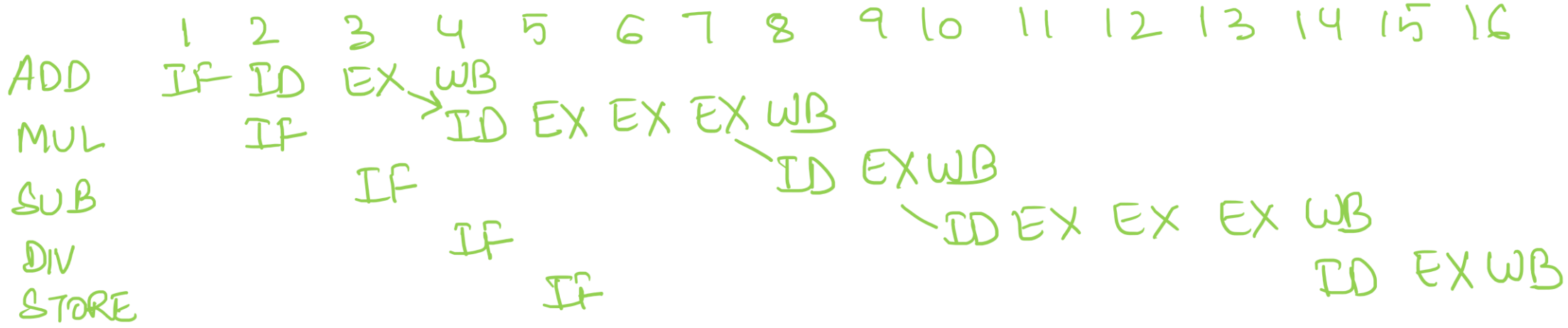
ADD	R5, R0, R1	; R5 ← R0 + R1
MUL	R6, R2, R5	; R6 ← R2 * R5
SUB	R5, R3, R6	; R5 ← R3 - R6
DIV	R6, R5, R4	; R6 ← R5/R4
STORE	R6, X	; X ← R6

A. 10

B. 12

C. 14

D. 16



Recent GATE Questions (2024 CS 1 mark)

An instruction format has the following structure: Instruction Number: Opcode destination reg, source reg-1, source reg-2

Consider the following sequence of instructions to be executed in a pipelined processor:

I1: DIV R3, R1, R2
I2: SUB R5, R3, R4
I3: ADD R3, R5, R6
I4: MUL R7, R3, R8

RAW

Which of the following statements is/are TRUE?

A. There is a RAW dependency on R3 between I1 and I2
B. There is a WAR dependency on R3between I1 and I3
C. There is a RAW dependency on R3 between I2 and I3
D. There is a WAW dependency on R3 between I3 and I4

2024 CSE 1 Mark

- Consider a 5-stage pipelined processor with Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory Access (MEM), and Register Writeback (WB) stages. Which of the following statements about forwarding is/are CORRECT?

A. In a pipelined execution, forwarding means the result from a source stage of an earlier instruction is passed on to the destination stage of a later instruction.

B. In forwarding, data from the output of the MEM stage can be passed on to the input of the EX stage of the next instruction.

C. Forwarding cannot prevent all pipeline stalls.

D. Forwarding does not require any extra hardware to retrieve the data from the pipeline stages

2024 CSE 1 mark

- Consider a 3-stage pipelined processor having a delay of 10 ns (nanoseconds), 20 ns, and 14 ns, for the first, second, and the third stages, respectively. Assume that there is no other delay and the processor does not suffer from any pipeline hazards. Also assume that one instruction is fetched every cycle. The total execution time for executing 100 instructions on this processor is _____ ns.

no. of stages $(m) = 3$

no. of tasks $(n) = 100$

$T_p = \max(10, 20, 14)$

$= 20$

$\therefore$ execution time $= 3 \times 20 + 99 \times 20$

$= 2040$ ns

2023 CSE 1 mark

For a pipelined CPU with a single ALU, consider the following situations Memory Interfacing

1. The j+ 1 instruction uses the result of the j-th instruction as an operand
2. The execution of a conditional jump instruction.
3. The j-th and j + 1 instruction require the ALU at the same time.

Which of the above can cause a hazard?

ALL

Delay Due to Branch Instructions

- A branch disrupts the normal flow of control. If an instruction is a branch instruction (which is known only at the end of instruction decode step), the next instruction may be either the next sequential instruction (if the branch is not taken) or the one specified by the branch instruction (if the branch is taken).
- Unconditional branches (always taken)
- Conditional branches (may or may not be taken)

Delay Due to Branch Instructions

- In our pipeline the maximum ideal speed up is 5. Let the percentage of unconditional branches in a set of typical programs be 5% and that of conditional branches be 15 %. Assume that 80% of the conditional branches are taken in the programs. Assume that n instructions are there which take one cycle each to complete (ideal case). What is the %age of loss of speed up due to branches?(Branch taken: delay of 3 cycles, branch not taken: delay of 2 cycles)

Sol. No. of cycles per inst^n = 1 (ideal case)

Avg delay cycles due to unconditional branches = $3 \times 0.05 = 0.15$

Avg. delay due to conditional branches =

$= 3*(0.15*0.8) + 2*(0.15*0.2)$

$= 0.36 + 0.06 = 0.42$

$\therefore$ Speed up $\bar{c}$ branches $= \frac{5}{1+0.15+0.42} \approx 3.18$

% age loss of speed up due to branches $= \frac{5-3.18}{5} \times 100 = 36.4\%$

Hardware Modification to Reduce Delay Due to Branches

- The primary idea is to find out the address of the next instruction to be executed as early as possible.
- If we put a separate ALU in the decode stage of the pipeline to find the result of condition, we can make a decision to branch or not at the end of DECODE stage.
- By adding this extra circuitry we have reduced delay to 1 cycle if branch is taken and to zero if it is not taken.

Hardware Modification to Reduce Delay Due to Branches

- Assume again 5% unconditional jumps, 15% conditional jumps and 80% of conditional jumps are taken. Assume that extra hardware is used to detect branching in the 2nd stage only. By adding this extra circuitry, we have reduced delay to 1 cycle if branch is taken and to zero if not taken. What is the %age of loss of speedup and improvement from previous case?

Average delay cycles c extra h/w

$= 1 * 0.05 + (0.15 * 0.8) * 1 + (0.15 * 0.2) * 0$

$= 0.17$

Gain of 22% from extra h/w

$\therefore$ speedup c branches $= \frac{5}{1.17} \approx 4.27$

% loss of speed up $= \frac{5 - 4.27}{5} \times 100 = 14.6\%$

Hardware Modification to Reduce Delay Due to Branches

- Commercial processors are more complex and have a variety of branch instructions. It may not be cost effective to add hardware.
- We will discuss two methods both of which depend on predicting the instruction which will be executed immediately after a branch instruction. The prediction is based on the execution time behavior of a program.
- The first method we discuss is less expensive in the use of hardware and consequently less effective. It uses a small fast memory called a branch prediction buffer to assist the hardware in selecting the instruction to be executed immediately after a branch instruction.
- The second method which is more effective also uses a fast memory called a branch target buffer. This memory, however, has to be much larger and requires more control circuitry.

Branch Prediction Buffer

- In this technique some of the lower order bits of the address of branch instructions in a program segment are used as addresses of the branch prediction buffer memory.

Address	*Contents*	*Prediction bits*
Low order bits of branch instruction address	Address where branch will jump	2 bits

Figure 3.16 The fields of a branch prediction buffer memory.

TABLE 3.4 How Branch Prediction Bits are Changed

Current prediction bits	00	00	01	01	10	10	11	11
Branch taken?	Y	N	Y	N	Y	N	Y	N
New prediction bits	01	00	10	00	11	01	11	10

Branch Prediction Buffer

- The contents of each location of this buffer memory is the address of the next instruction to be executed if the branch is taken.
- In addition, two bits are used to predict whether a branch will be taken when a branch instruction is executed.
- If the prediction bits are 00 or 01, the prediction is that the branch will not be taken.
- If the prediction bits are 10 or 11 then the prediction is that the branch will be taken. While executing an instruction, at the DE step of the pipeline, we will know whether the instruction is a branch instruction or not.
- If it is a branch instruction, the low order bits of its address are used to look up the branch prediction buffer memory. Initially the prediction bits are 00.

Branch Prediction Buffer

- The prediction bits are examined. If they are 10 or 11, control jumps to the branch address found in the branch prediction buffer. Otherwise the next sequential instruction is executed. Experimental results show that the prediction is correct 90% of the time. With 1000 entries in the branch prediction buffer, it is estimated that the probability of finding a branch instruction in the buffer is 95%. The probability of finding the branch address is at least $0.9 \times 0.95 = 0.85$.
- Why should there be 2 bits in the prediction field? Would it not be sufficient to have only one bit?
- A single bit predictor incorrectly predicts branches more often, particularly in most loops, compared to a 2-bit predictor. Thus, it has been found more cost-effective to use a 2-bit predictor.

Branch Target Buffer

- Unlike a branch prediction buffer, a branch target buffer is used at the instruction fetch step itself.
- Observe that the address field has the complete address of all branch instructions. The contents of BTB are created dynamically.
- When a program is executed whenever a branch statement is encountered, its address and branch target address are placed in BTB.
- Remember the fact that an instruction is a branch will be known only during the decode step. At the end of execution step, the target address of the branch will be known if branch is taken. At this time the target address is entered in BTB and the prediction bits are set to 01.

Address	*Contents*	*Prediction bits*
Address of branch instruction	Address where branch will jump	1 or 2 bits (optional)

Figure 3.17 The fields of a branch target buffer memory.

Branch Target Buffer

- Once a BTB entry is made, it can be accessed at instruction fetching phase itself and target address found.
- Typically when a loop is executed for the first time, the branch instruction governing the loop would not be found in BTB.
- It will be entered in BTB when the loop is executed for the first time.
- When the loop is executed the second and subsequent times the branch target would be found at the instruction fetch phase itself thus saving 3 clock cycles delay.
- Observe that we have to search BTB to find out whether the fetched instruction is in it. Thus, BTB cannot be very large. About 1000 entries are normally used in practice.

Branch Target Buffer

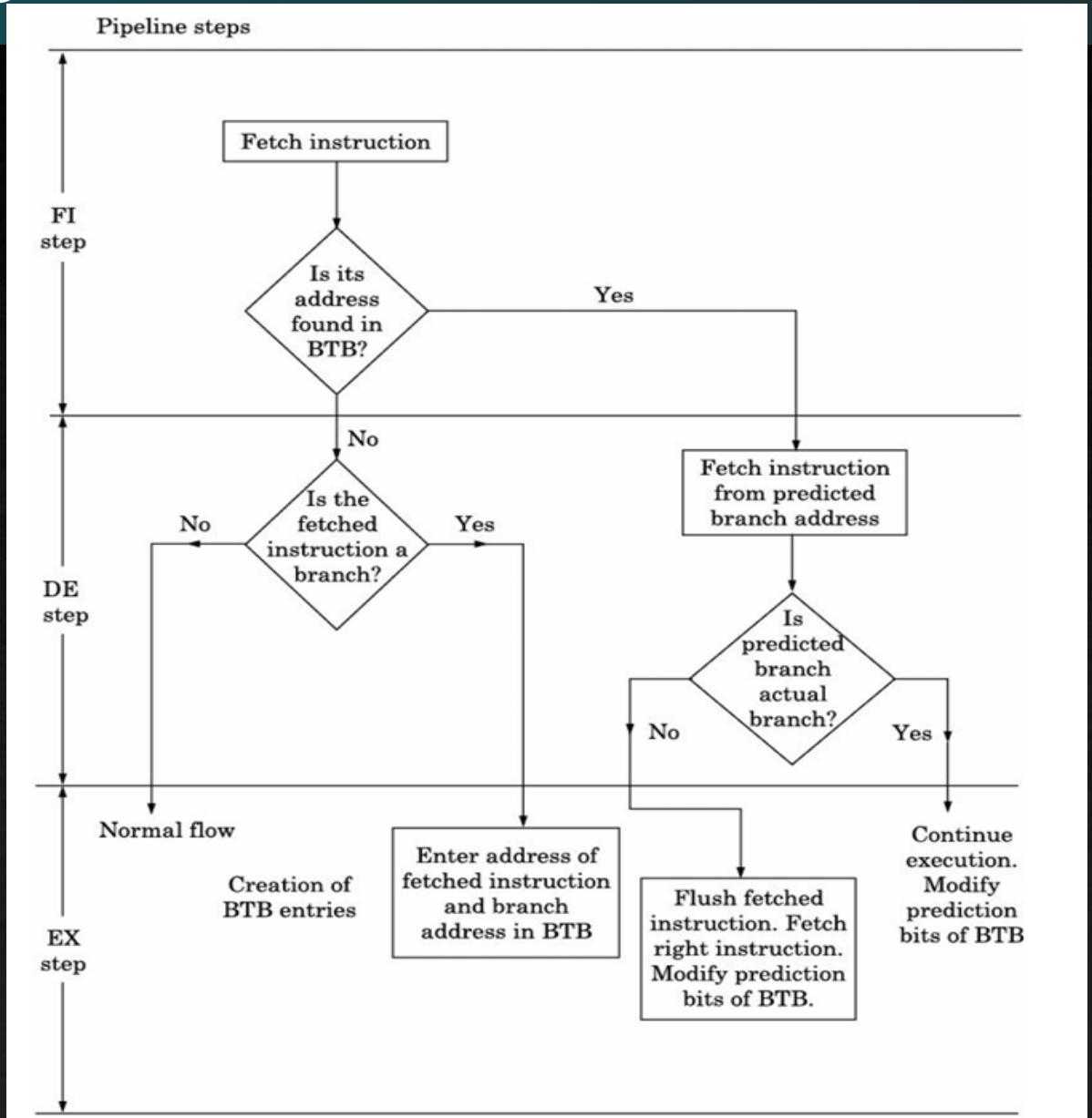


Figure 3.18 Branch Target Buffer (BTB) creation and use.

Branch Target Buffer

- Reduction in speedup due to branches when BTB is employed using the previous data.

Data: unconditional branches = 5% Conditional branches = 15%

Taken branches = 80% of conditional

Assumptions: Branch instns are found in BTB ∈ probability of 0.95. 90% cases - the branch prediction by BTB is correct. There is no special branch address calc. h/w.

→ Unconditional branches (UCB)

Avg delay when UCB are found in BTB = 0

Avg delay when UCB are not found in BTB = 3 ? 0.15

Avg. delay due to unconditional branches = 3 * 0.05 = 0.1

Branch Target Buffer

Avg delay due to conditional branches when they are found in BTB = 0

Avg delay when CB are not found in BTB = $3 * 0.8 + 2 * 0.2 = 2.8$

As prob. of not being in BTB = 0.05, the avg delay due to conditional branches = $0.05 \times 2.8 = 0.14$

Avg delay due to misprediction of conditional branches when found in BTB = $0.1 \times 2.8 \times 0.95 = 0.2666$

As 5% are UCB & 15% are CB, The avg. delay due to branches = $0.05 \times 0.15 + 0.15(0.14 + 0.266) = 0.0684$

∴ speed up Σ branches when BTB is used = $\frac{5}{1 + 0.0684} \cong 4.67$

Branch Target Buffer

% loss of speed up due to branches $= \frac{5 - 4.67}{5} \times 100$

$= 6\%$

GATE CSE 2024 2 marks

- A non-pipelined instruction execution unit operating at 2 GHz takes an average of 6 cycles to execute an instruction of a program P. The unit is then redesigned to operate on a 5-stage pipeline at 2 GHz. Assume that the ideal throughput of the pipelined unit is 1 instruction per cycle. In the execution of program P, 20% instructions incur an average of 2 cycles stall due to data hazards and 20% instructions incur an average of 3 cycles stall due to control hazards. The speedup (rounded off to one decimal place) obtained by the pipelined design over the non-pipelined design is _____
- Ans : 3.0

GATE CSE 2022 2 marks

- A processor X_1 operating at 2 GHz has a standard 5-stage RISC instruction pipeline having a base CPI (cycles per instruction) of one without any pipeline hazards. For a given program P that has 30% branch instructions, control hazards incur 2 cycles stall for every branch. A new version of the processor X_2 operating at same clock frequency has an additional branch predictor unit (BPU) that completely eliminates stalls for correctly predicted branches. There is neither any savings nor any additional stalls for wrong predictions. There are no structural hazards and data hazards for X_1 and X_2. If the BPU has a prediction accuracy of 80%, the speed up (rounded off to two decimal places) obtained by X_2 over X_1 in executing P is _____.

- Ans : 1.43

Gate CS 2020 2 marks

- Consider a non-pipelined processor operating at 2.5 GHz. It takes 5 clock cycles to complete an instruction. You are going to make a 5-stage pipeline out of this processor. Overheads associated with pipelining force you to operate the pipelined processor at 2 GHz. In a given program, assume that 30% are memory instructions, 60% are ALU instructions and the rest are branch instructions. 5% of the memory instructions cause stalls of 50 clock cycles each due to cache misses and 50% of the branch instructions cause stalls of 2 cycles each. Assume that there are no stalls associated with the execution of ALU instructions. For this program, the speedup achieved by the pipelined processor over the non-pipelined processor (round off to 2 decimal places) is _____.

- Ans : 2.16

GATE CS 2016 2 marks

- Suppose the functions F and G can be computed in 5 and 3 nanoseconds by functional units U_f and U_g, respectively. Given two instances of U_f and two instances of U_g, it is required to implement the computation F (G (X_1)) for 1< i < 10. Ignoring all other delays, the minimum time required to complete this computation in nanoseconds is …….

- Ans : 28ns

GATE 2014 CS 2 marks

- Consider a 6-stage instruction pipeline, where all stages are perfectly balanced. Assume that there is no cycle-time overhead of pipelining. When an application is executing on this 6-stage pipeline, the speedup achieved with respect to non-pipelined execution if 25% of the instructions incur 2 pipeline stall cycles is_____.

- Ans = 4.0

GATE CSE 2013 2 marks

- Consider an instruction pipeline with five stages without any branch prediction: Fetch Instruction (FI), Decode Instruction (DI), Fetch Operand (FO), Execute Instruction (El) and Write Operand (WO). The stage delays for FI, DI, FO, El and WO are 5 ns, 7 ns, 10 ns, 8 ns and 6 ns, respectively. There are intermediate storage buffers after each stage and the delay of each buffer is 1 ns. A program consisting of 12 instructions I1, I2, I3, …., I12 is executed in this pipelined processor. Instruction I4 is the only branch instruction and its branch target is I9. If the branch is taken during the execution of this program, the time (in ns) needed to complete the program is…….

- Ans : 165 ns (15 cycles x 11 ns)

GATE CS 2016 2 marks

- Consider a 3 GHz (gigahertz) processor with a three stage pipeline and stage latencies T1,T2 and T3 such that T1 = 3T2/4 = 2T3. If the longest pipeline stage is split into two pipeline stages of equal latency , the new frequency is _____ GHz, ignoring delays in the pipeline registers.

- Ans : 4 GHz

Practice Question

- In our pipeline the maximum ideal speedup is 5. Let the percentage of unconditional branches in asset of typical programs be 7% and that of conditional branches be 18%. Assume that 85% of the conditional branches are taken in the program. (Branch taken delay = 3 cycles and not taken = 2 cycles)
  a. What is the %age loss of speedup due to the branches?
  b. Extra hardware is used at decode stage to reduce delay cycles to 1 for a branch being taken and 0 cycles for branch being not taken. What is the %age loss of speedup from ideal case and gain over the previous one?
  c. A BTB is used with a prediction accuracy of 92% and the probability of .95 that a branch entry would be present in BTB. What is the %age loss of speedup from ideal case?

Software Method to Reduce Delay Due to Branches

- The primary idea is for the compiler to rearrange the statements of the assembly language program in such a way that the statement following the branch statement (called a *delay slot*) is always executed once it is fetched without affecting the correctness of the program.
- This may not always be possible but analysis of many programs shows that this technique succeeds quite often.

Software Method to Reduce Delay Due to Branches

- Example:

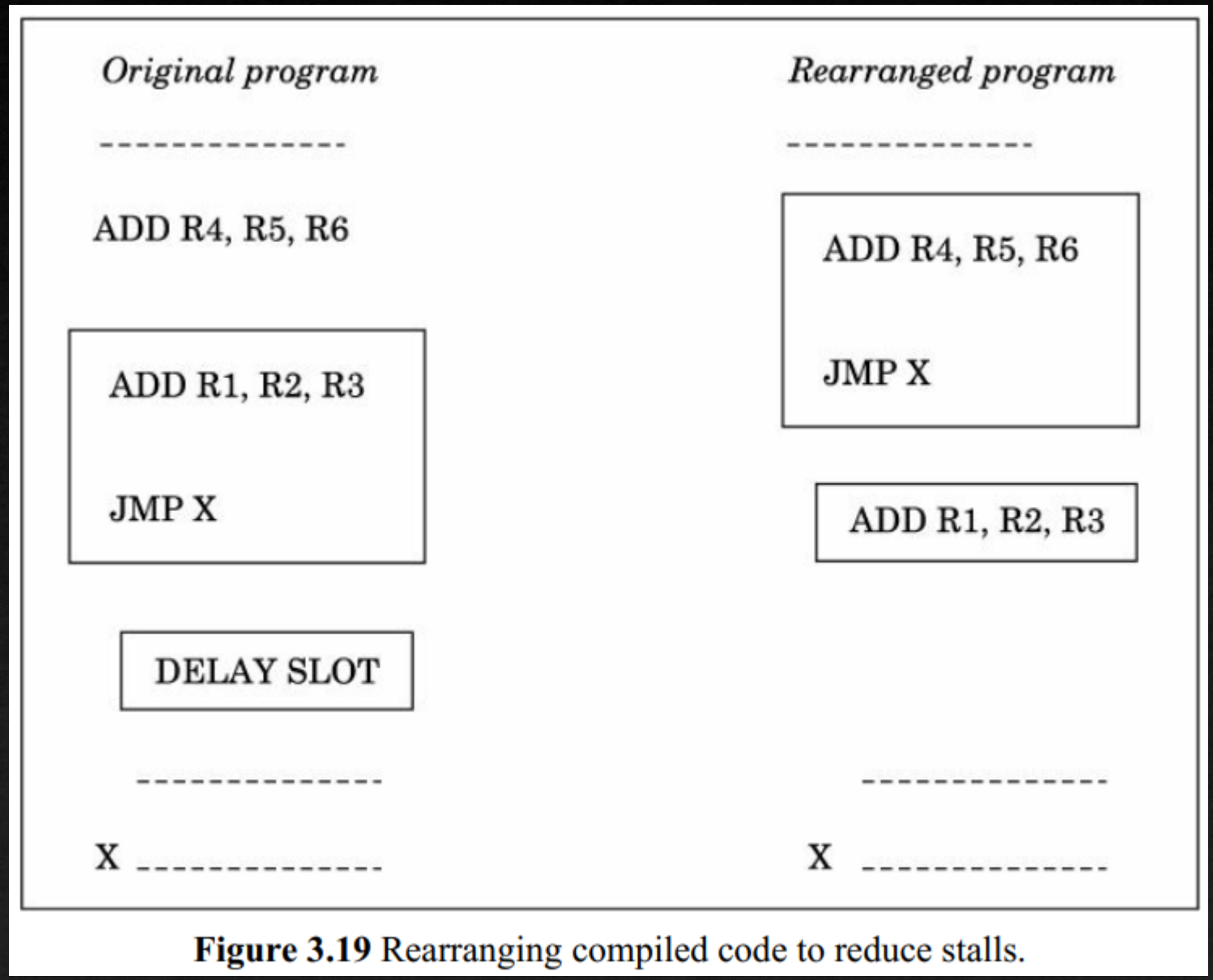


Figure 3.19 Rearranging compiled code to reduce stalls.

Software Method to Reduce Delay Due to Branches

- Example: Observe that the delay slot is filled by the target instruction of the branch.
- If the probability of the branch being taken is high then this procedure is very effective.

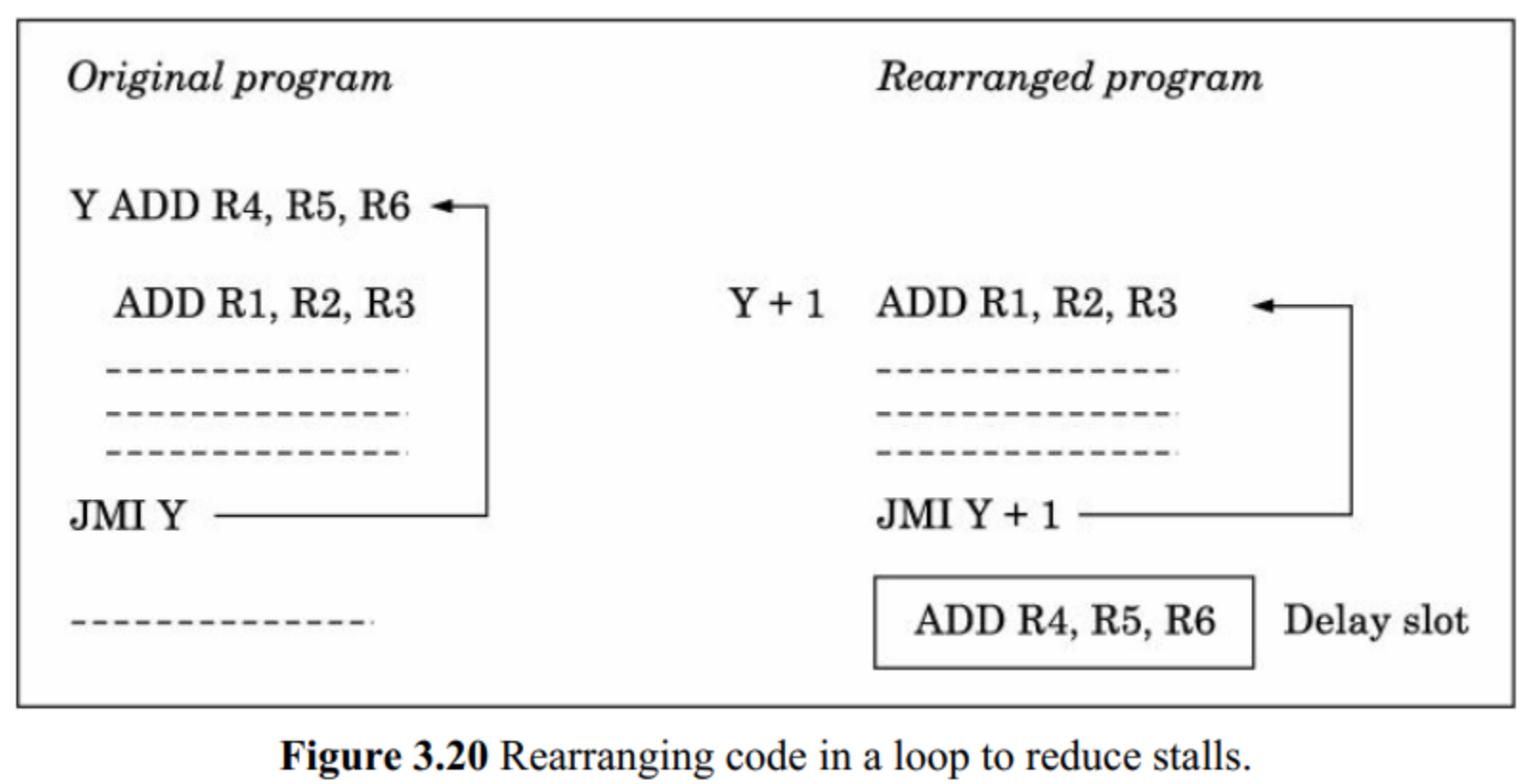


Figure 3.20 Rearranging code in a loop to reduce stalls.

Software Method to Reduce Delay Due to Branches

- Forward Branching

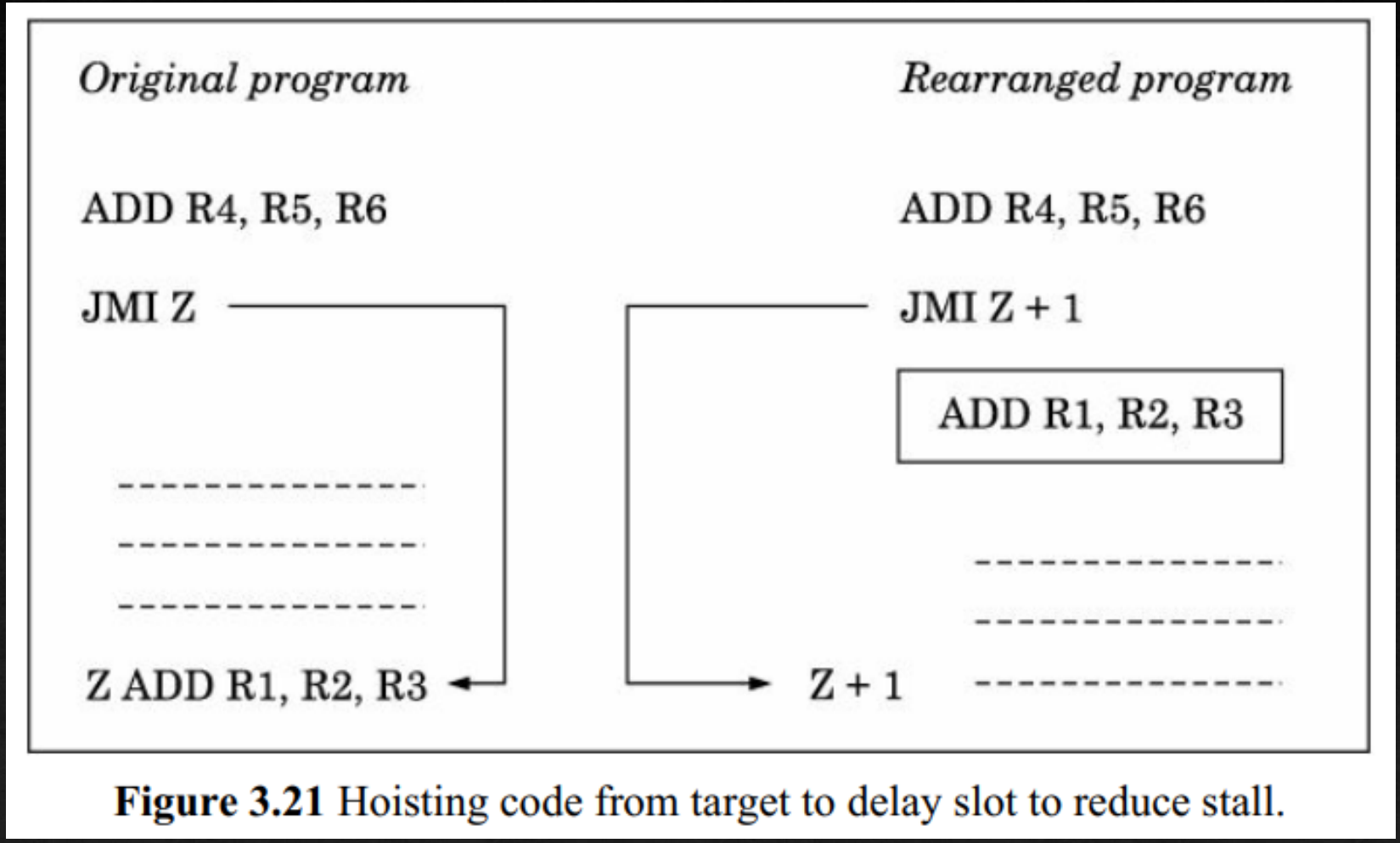


Figure 3.21 Hoisting code from target to delay slot to reduce stall.

Software Method to Reduce Delay Due to Branches

- Loop Unrolling
- There are many compiler optimizations techniques to speed up execution of loops and loop unrolling is one such compiler optimization technique.

```
for i in range(5):
    print(i)
```

```
i = 0
exit_loop = i < 5
loop_body:
    print(i)
    i += 1
    exit_loop = i < 5
    if exit_loop:
        jump loop_body
end_loop:
    ...
```

```
print(0)
print(1)
print(2)
print(3)
print(4)
```

Loop Unrolling

- Loop unrolling is a program transformation that trades code size for execution speed.
- The basic way it operates is that if we have (for the sake of a simple example) a loop that assigns 16 array elements thus:

```
1  for ( int i=0; i<16; i++ )
2      data[i] = i;
```

- it can be 'unrolled' by instantiating the loop body twice, and increasing the stride of the induction variable i, like so:

```
1  for ( int i=0; i<16; i+=2 )
2  {
3      data[i] = i;
4      data[i+1] = i+1;
5  }
```

Loop Unrolling

- This can improve instruction cache utilization, expose more instruction-level parallelism, and make the job of any prefetching logic a little easier.

```
1  for ( int i=0; i<16; i+=4 )
2  {
3      data[i] = i;
4      data[i+1] = i+1;
5      data[i+2] = i+2;
6      data[i+3] = i+3;
7  }
```

Difficulties in Pipelining

- Difficulty due to interruption of normal flow of a program due to events such as illegal instruction codes, page faults, and I/O calls. We call all these *exception conditions.*

TABLE 3.5 Exception Types in a Computer

Exception type	*Occurs during pipeline stage?*		*Resume or terminate*
	Yes/No	*Which stage?*	
I/0 request	No		Resume
OS request by user program	No		Resume
User initiates break point during execution	No		Resume
User tracing program	No		Resume
Arithmetic overflow or underflow	Yes	EX	Resume
Page fault	Yes	FI, MEM	Resume
Misaligned memory access	Yes	FI, MEM	Resume
Memory protection violation	Yes	FI, MEM	Resume
Undefined instruction	Yes	DE	Terminate
Hardware failure	Yes	Any	Terminate
Power failure	Yes	Any	Terminate

Difficulties in Pipelining

- *precise exceptions*
- If the pipeline processing can be stopped when an exception condition is detected in such a way that all instructions which occur before the one causing the exception are completed and all instructions which were in progress at the instant exception occurred can be restarted (after attending to the exception) from the beginning, the pipeline is said to have *precise exceptions*.

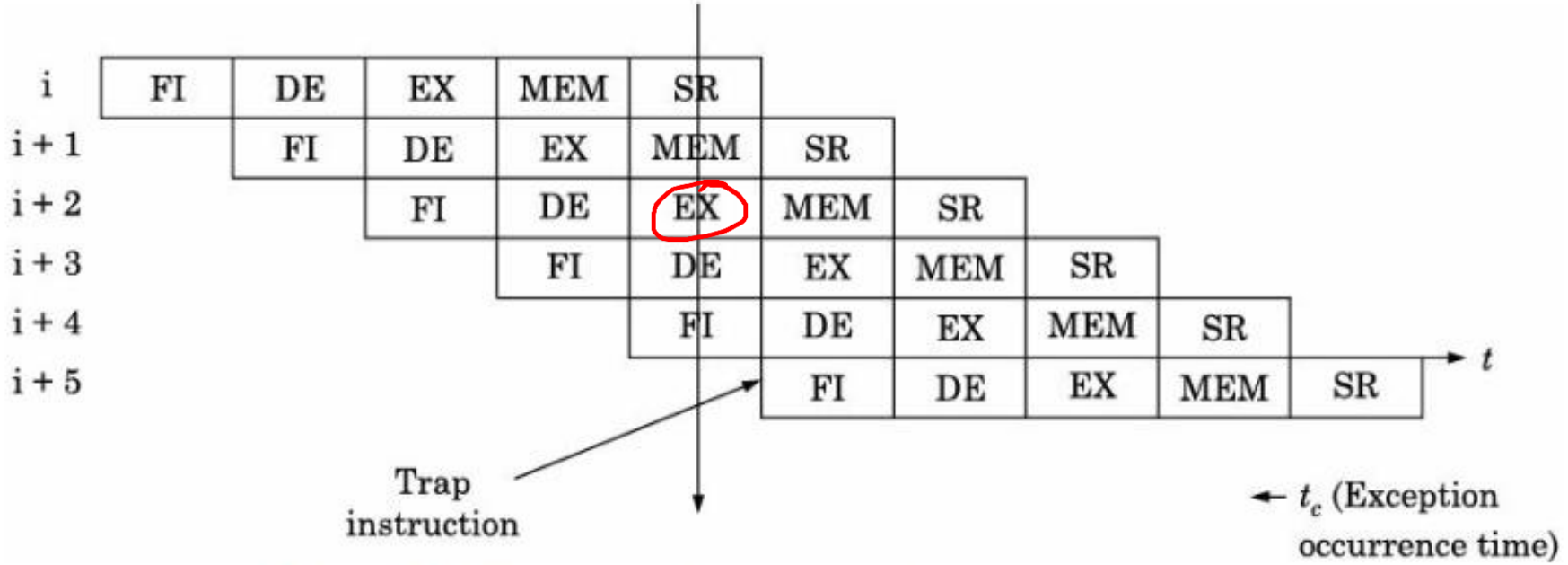


Figure 3.22 Occurrence of an exception during pipeline execution.

Exception handling

- When an exception is detected, the following actions are carried out:

1. As soon as the exception is detected turn off write operations for the current and all subsequent instructions in the pipeline [Instructions (i + 2), (i + 3) and (i + 4) in Fig.3.22].

2. A trap instruction is fetched as the next instruction in pipeline (Instruction i + 5 in Fig. 3.22).

3. This instruction invokes OS which saves address (or PC of the program) of faulting instruction to enable resumption of program later after attending to the exception.

Superpipelining

- In a pipelined processor we assume that each stage of pipeline takes the same time, namely, one clock interval.
- In practice some pipeline stages require less than one clock interval. For example, DE and SR stages normally may require less time.
- Thus, we may divide each clock cycle into two phases and allocate intervals appropriate for each step in the instruction cycle.
- if the steps are subdivided into two phases such that each phase needs different resources thereby avoiding resource conflicts, pipeline execution can be made faster as shown in Fig. 3.23. This method of pipeline execution is known as *superpipelining.*
- In the steady state one instruction will take half a clock cycle under ideal conditions. All the difficulties associated with pipeline processing (namely various dependencies) will also be there in superpipelined processing.
- As superpipelined processing speeds up execution of programs, it has been adopted in many commercial high performance processors such as MIPS R4000.