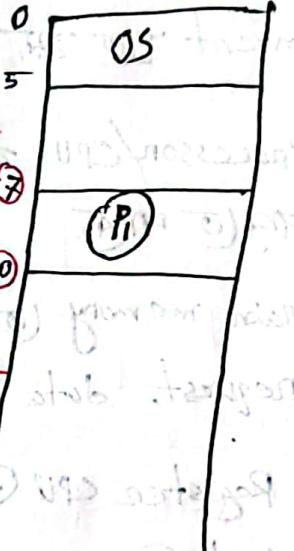


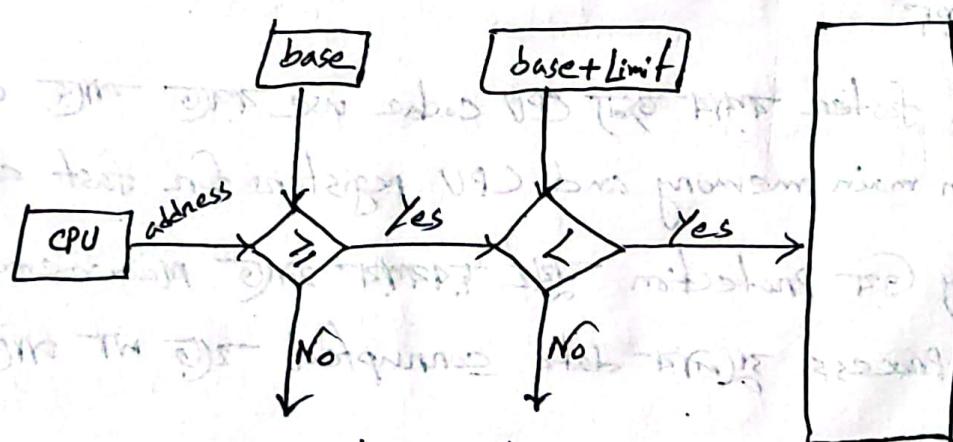
Main Memory

- * Program হাতে main memory তে load কৰা হ'লে process ক নিয়ে প্রক্রিয়া করতে পারে।
- * Processor/CPU আবশ্যিক Main Memory and Registers ক direct access করতে পারে।
- * Main memory কে 24-bit address দিয়ে আবশ্যিক address, 64-bit address প্রদাতে read request, data and write request করা যাব।
- * Register CPU কে ধূৰ করতে পারে, CPU আবশ্যিক Register কে 24-bit Data কে access করতে পারে, একটি CPU clock কা প্রয়োজন করে কাম সম্ভব।
- * Main memory কে access করতে প্রসেসরে CPU কে কাটা করে cycle maintain করতে হয়, এই stall করিয়ে, stall করে access করতে হ'ল debug করা কুরআন।
- * Access faster করাব জন্য CPU cache কে ব্যবহার করা যাব, Cache sits between main memory and CPU registers for fast access.
- * Memory protection করাব করাব জন্য Main memory কে process করলেখ করে করে data corrupted হ'ল না পারে।

#Base and Limit Registers

- * A pair of base and limit registers define the logical address space.
- * Each process RAM কে 256 MB Address Space কে allocation করে করে 24-bit address space কে access করতে পারব, এটা করে করে protection of the memory. 64-bit protection কি- maintain করতে Base and Limit Registers কে ব্যবহার করা যাব।

- * Does not process main memory (କେବଳ address space allocation)
 - Process Limit** (Limit of the process.
 - $\text{Limit Register} = \text{Base} + \text{Limit} - 1$
 - * CPU address space generate 2³², 32-bit address space of Process
 - Base & Limit Register କେବଳ 32-bit
 - Address generated by Base + Address
 - Base & Limit Register - To check address generated by CPU
 - Is the address generated by CPU in Main Memory or not?
 - 
 - $$\boxed{\text{Base} \leq \text{Generated Address} \leq \text{Limit Register}}$$
 - Valid for access of Main memory.



Trap to operating system monitor -
Addressing error

Address Space

System Address Space \rightarrow বিভিন্ন প্রক্রিয়া,

→ Logical Address Space

→ Physical Address Space

প্রিমিয়ার Logical Address Space Separate Physical Address Space

এবং উভয় bound \rightarrow সম্পর্ক,

Logical Address Space \rightarrow কোন Process generate \rightarrow আর আরও

আরও CPU যাইছে Local/Logical Address Space provide আর

জুড়ে অস্থায়ে process এবং Data সুলভ আর আর, Multiple
process এবং Logical Address space same জুড়ে আরও আরও

(এটা) Local address. এটা Logical Address Space \rightarrow Virtual

Address কোন আর, Process এবং জুড়ে কথাই Logical Address

আরও এটি জুড়ে কথাই Address space এবং allocation Main
memory \rightarrow আরও,

Physical Address Space \rightarrow কোন Process main memory \rightarrow (এই)

কথাই - Address space এবং allocation \rightarrow আরও জুড়ে \rightarrow Physical

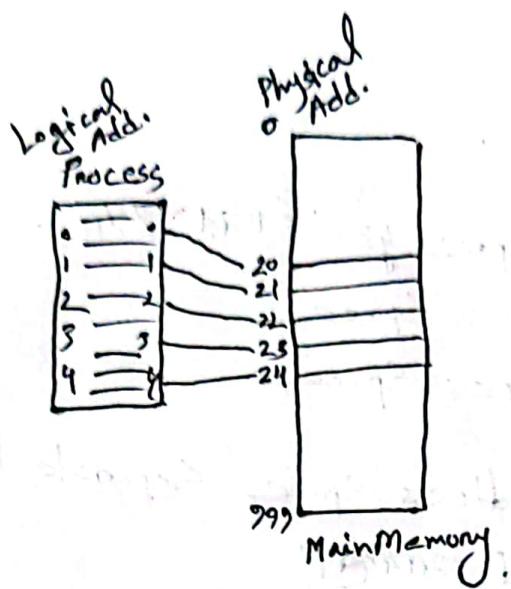
Address. কোন Process এবং প্রিমিয়ার Physical Address \rightarrow আর-

corresponding Logical Address \rightarrow সুলভ - আরও mapped \rightarrow আরও,

Process সুলভ - Physical Address কথাই same \rightarrow আরও,

কোন Process Ready state এ আরও আরও - একই logical and
physical address same \rightarrow আরও,

Logical Address and physical addresses are the same in compile-time
and load-time address-binding schemes. and they are different
in execution-time address-binding scheme.



Logical Address space generate
by CPU, Main Process or
Compiler

Physical Address space
generate by Main process or Run

Memory Management Unit (MMU)

MMU is a hardware device that maps logical address to corresponding physical address mapped by MMU.

Logical Address to Physical Address mapped by MMU

MMU Scheme follow flat, segmented simplest scheme

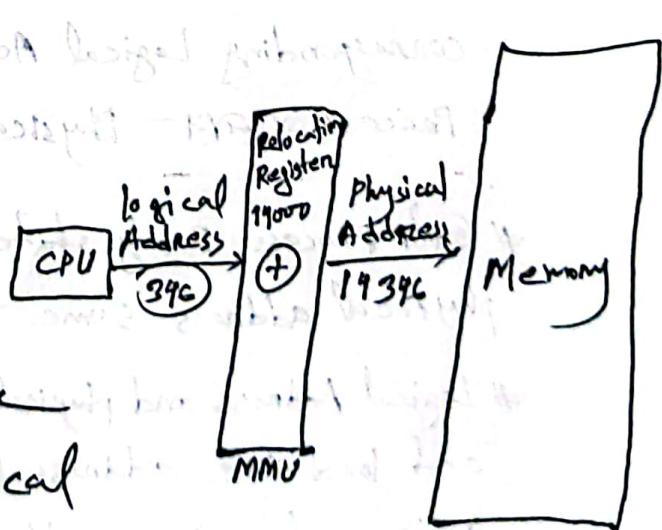
Flat Base register/Relocation register + value

Flat Physical Address - MS-DOS on Intel 8086

Segmented scheme follow 32-bit CPU; 4 relocation register used

User Program deals with logical Address, it never sees the real physical Address.

All the logical addresses are bound to corresponding physical Address



Dynamic Loading

- * RAM এর capacity দ্বাৰা বেতন কোৱা process আৰি execute কৰিব।
ইয়ে কৈমান কৰি deal কৰতে dynamic loading use কৰা হৈ।
 - * কোৱা process এবং task পুলেকে multiple sub-task কৰিব।
ইয়ে, এন্টি sub-task এবং এন্টি group কৰি বলি দ্বাৰা routine.
Usually এন্টি routine এবং size same হৈ, size এবং এন্টি
base এন্টি routine কৰা হৈ, এবং size এবং equivalent to
2^n. এছতে RAM এবং capacity দ্বাৰা বেতন size এবং process
পুলেক এবং একটি part কৰি execution কৰি প্ৰিমেজ প্ৰোগ্ৰাম
একটি part কৰি routine কৰি allocation কৰি RAM কৰিব।

- * কোৱা process এবং routine কৰা কৰিব।

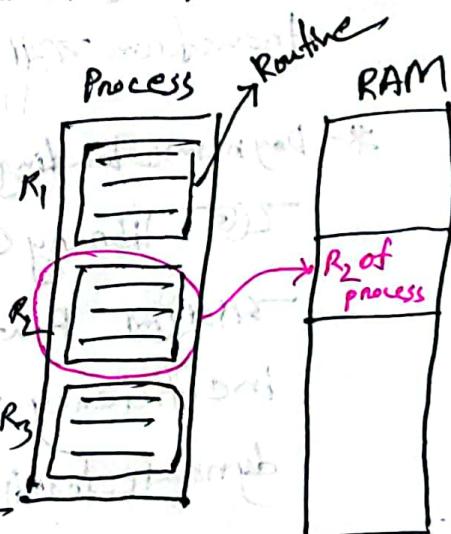
Load কৰা হৈলি RAM এ যাওয়া হৈ।
নি পুলি routine কৰি call কৰা হৈ।

- * দ্বাৰা routine কৰি প্ৰিমেজ লাই দ্বাৰা

routine কৰি কাশৰহী RAM কৰি load কৰা।
হৈলি,

- * আৰি routine কৰি relocate কৰা হৈ।

দেখি GDB disk কৰি relocatable load
format কৰা হৈ, Basically executable
file। দ্বাৰা disk কৰি।



- * এখন large amount of code very often or frequently
execute কৰাৰ প্ৰিমেজ লাই কৰি কৈসে কৈসে useful.

- * এই dynamic loading কৰি পুলি procedure কৰি OS কৰি কৈসে
special support এবং প্ৰিমেজ লাই কৰি। Dynamic loading কৰি proper
mechanism কৰি function পুলেক কৰি implement কৰা হৈ।
Programmer program কৰি design কৰি কৈসে,
OS dynamic loading implement কৰাৰ কৈসে necessary library support

নির্মাণ করে provide করে।

Dynamic Linking

* কোর্ট Program কে দুটি Part \rightarrow header এবং body. প্রথম Instructions দুটি Program এ বস্তুতই কোর্ট instructions তাজে execute করে যেখানে necessary library support লাগে। রয়েছে header এবং body part sync করে। executable file create করে, তবে executable file কিন্তু execute করা হয়।

* অবশ্য System library এর Program code কে combine করে।

বাস্তবে Runtime environment এর loader কে ধরে। loader কে ধরে। এটি কোর্ট executable file কে execution করে। binary program image কে transform করে। এটি কোর্ট বলে একই static linking.

* Dynamic loading করে। এক্ষেত্রে process কে কোর্ট Routine execute করে। library এসেওর lets say 2000, কিন্তু library function শুধুমাত্র Routine কে। আর 16 bit include করা আছে। এক্ষেত্রে inefficiency হয়। অবশ্য, static linking কে কোর্ট issues করে। dynamic loading করে। এক্ষেত্রে face করতে হবে, কোর্ট dynamic linking use করতে হবে।

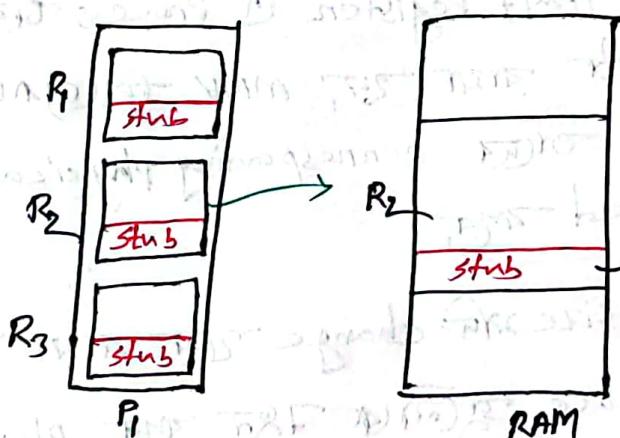
Dynamic linking করে। এক্ষেত্রে linking or Postponed বলা হয়। execution time start হবে। আগে পড়ে।

* Stub নাম করে। small portion of code প্রতিবেদন। Routine কে 2000 কে। এটি কোর্টের particular Routine কে ধরে। library কে। এটি। কোর্টের particular routine কে। এটি। কোর্টের particular routine কে। import করে।

* Stub কিভাবে Routine কে। Address দ্বারা replace করে। Routine কে। এটি। execute করতে হবে।

OS check करते हैं Routine के process के Memory Address (GCR) की तरफ, जिसका process के Memory Address space (GCR) Routine के Add करते हैं।

* Dynamic Linking libraries द्वारा use कराया जाने वाला एक ग्रेटर helpful. ~~एक~~ libraries को प्रोग्राम में Multiple Routine G Shared → 21725 217 System के लिए आवश्यक,



→ R₂ को execute करते हैं तथा Necessary libraries को RAM में load करते हैं। Address को load करने के लिए GCR R₂ GCR instructions load करते हैं। तो GCR को Replace करते हैं, तो यह बदलता है।

Contiguous Allocation

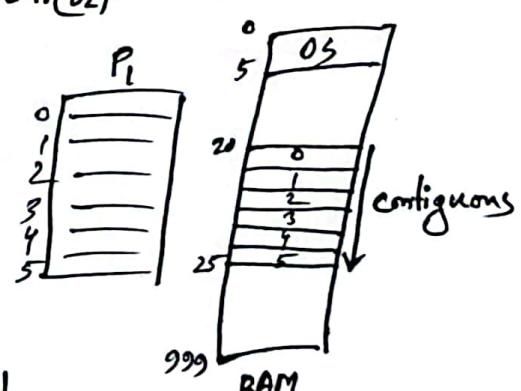
कोई Process के लिए RAM के execution के लिए allocation की जाती है तथा GCR के Memory के Address Space की भी जाती है जो लगभग अपेक्षित रूप से Contiguously होती है।

Main memory को OS and user process द्वारा accommodate होती है।

Resources Limited की allocation efficiently होती है।

Contiguous allocation is one of the early method.

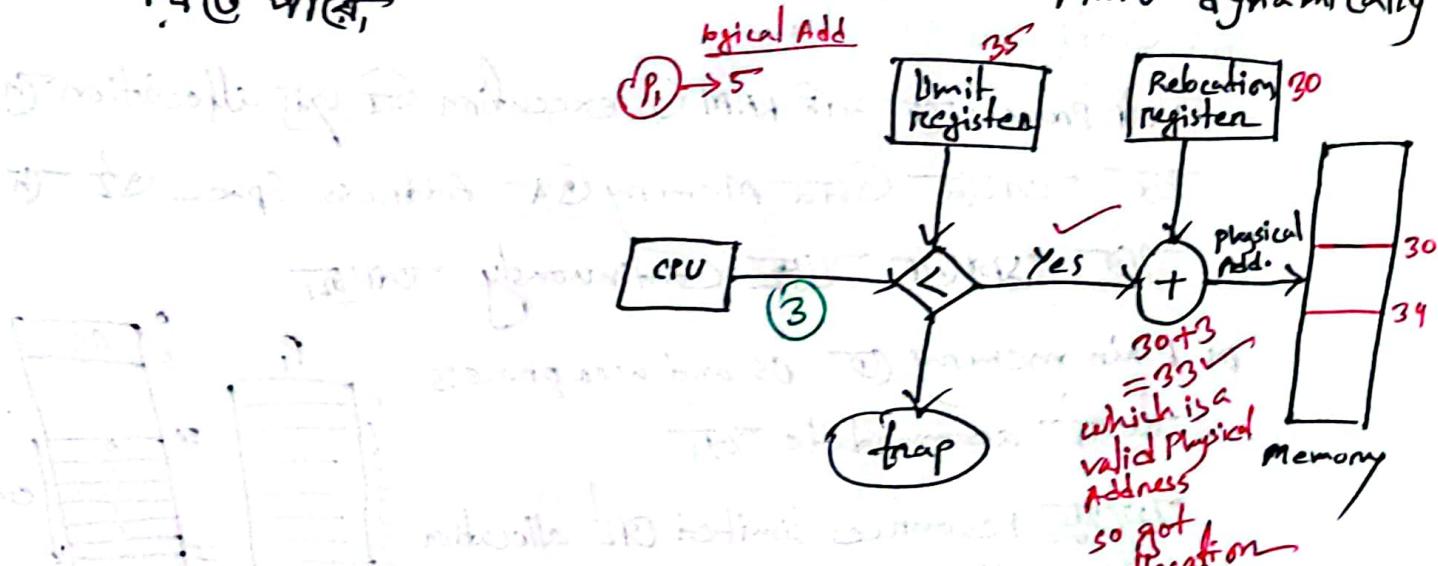
Main memory usually 2 or partition के divide करता है। lower address space को Resident OS को allocate करता है, जबकि interrupt vector वे OS के allocation के द्वारा होते हैं। इसके लिए OS कीिति-विभाजन decision नीति आवश्यक है।



OS यात्रु का allocation निम्नों RAM द्वारा या for data to different Part
ये RAM द्वारा योग्य अस्थि गणना करते/Address space का user
processes allocation करते

Relocation register/ Base register Process का allocation main
memory के ये address space के लिए start की की value का
store करते हैं, Limit Register का process का logical address
space का limit होता है, MMU के लिए logical address
के Run time के लिए corresponding physical address space
के आधर मapped होते हैं,

Kernel code का size बदले change के समय द्वारा
logical Address space का limit का physical address
space का आधर Mapped होते हैं जब तक MMU dynamically
देखते होंगे,

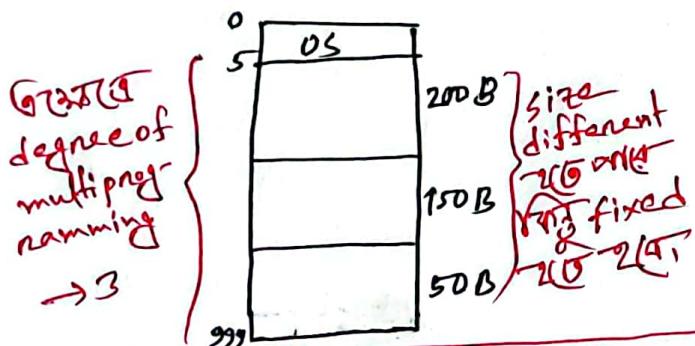


#Multiple-Partition Allocation

- RAM (Process allocation) द्वारा यह सभी विभिन्न Space जूले खोला जाता है।
- अनेक विभाजित Partition (divide करा द्यता, (2) Partition विभाजित करा द्यता)
- एक विभाजित करा द्यता

Fixed-sized partition

- RAM (process allocation) द्वारा यह (1) space जूले available - 200B, 150B, 50B
- fixed size (2), fixed number of partition & divide विभाजित करा द्यता



fixed-sized partition - 200B degree of multiprogramming Control द्यता,

- अनेक Partition (almost) द्यता process द्यता allocate द्यता याएं।

Drawback of fixed-sized partition

In this case,
Total Amount of
Hole → 50 + 20 + 40
= 110B

(1) Hole द्यता
विभाजित space जूले

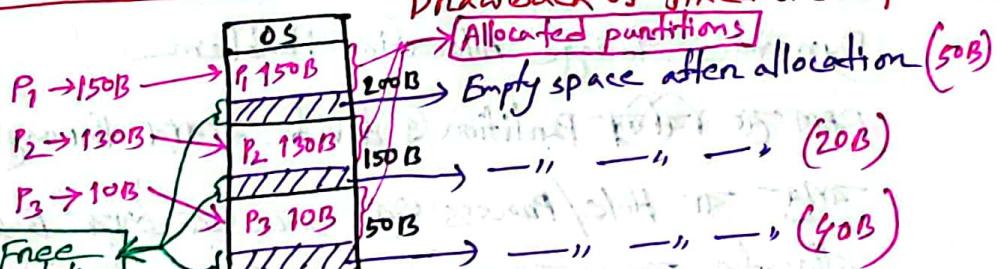
प्रति contiguous

रासायनिक RAM (

बहुत कम process

- कमज़ोर (1) Hole द्यता

Space Allocation नहीं।



fixed-sized partition द्यता द्योषित RAM द्यता (यह)
- कमज़ोर process द्यता allocated द्यता कमज़ोर द्यता
- बहुत द्यता allocated partition - कमज़ोर (यह)
- कमज़ोर process allocated द्यता कमज़ोर द्यता

This is the major drawback
of fixed-sized partitions.

Hole.

(1) Problem is - improve करा द्यता विभाजित

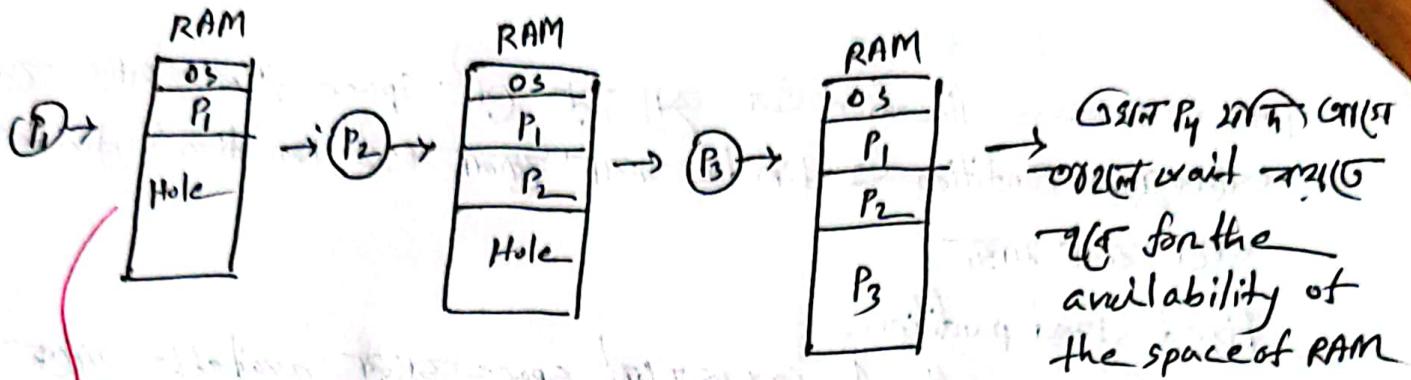
Variable partition introduce द्यता 200B,

Variable Partition

(1) Number of partitions fixed द्यता, अनेक Partition द्यता size

- equivalent to the size of process. कमज़ोर process RAM (

- कमज़ोर द्यता कमज़ोर Partition द्यता द्यता,



ପରିଷତ୍ତ P. ପର
 ଅନ୍ତରୀଳ Hole
 ଏବଂ-ଏବଂ ଏବଂ Combined
 ଏବଂ ପରିଷତ୍ତ ପରିଷତ୍ତ

Dynamic Memory allocation

Dynamic Storage Allocation Problem

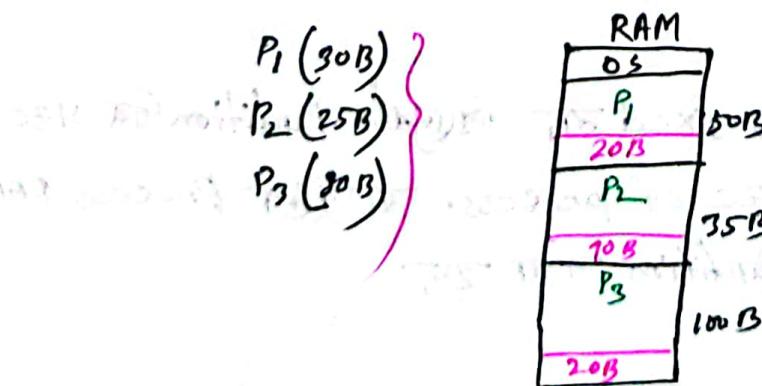
RAM টেক বাইট - Partition G divide রামার through ৭০ Hole G-এর টেক Base

মাপ - বা Hole/Process এর size কে প্রেরণ Base করা Process শূলকে
allocate করা হচ্ছে। এবং এই একটি পদ্ধতি algorithms use করা হচ্ছে। এই
ধরনের Algorithms এর নাম হচ্ছে -

First-fit, Best fit, Worst-fit
 Greedy algorithm এখনো Case G better perform কৰে, কেন্তব্যে
 diff directly most efficient কৰে যাবলৈ.

First-Fit Algorithm

Process ने इस Partition का Hole G' allocate करके
Hole G को छोटा करके 2nd Hole G'' बनाया है। यह Hole G' को Process ने कर
allocate किया है क्योंकि यह large enough.

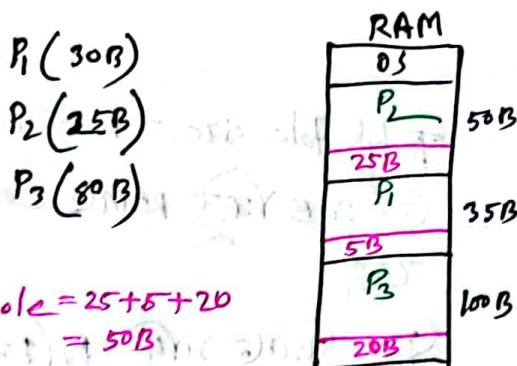


$$\text{hole} \rightarrow 20 + 10 + 20 = 50\Omega$$

Q3) Explain the algorithm to degree of multiprogramming controlled by M.

Best-Fit algorithm

ମୋଟ କାଣ୍ଡର ହୋଲ ଓ ପ୍ରୋସେସ ଏବଂ allocation କିମ୍ବା ହୁଏ ମୋଟ ପ୍ରୋସେସ ଏବଂ
allocation ଦେଖିଯାଇ ପାଇଁ ଡାକ୍ତର୍ମୁନ୍ ମଧ୍ୟେ smallest hole କ୍ରେଟେ କରିବାକୁ



$$\text{Hole} = 25 + 5 + 20 \\ = 50 \text{ B}$$

For P₁(30B)

$$\begin{array}{r} 50 - 30 = 20 \\ 35 - 30 = 5 \\ 100 - 30 = 70 \end{array}$$

$E_{\alpha} P_2 (80B)$

$$100 - 80 = 20$$

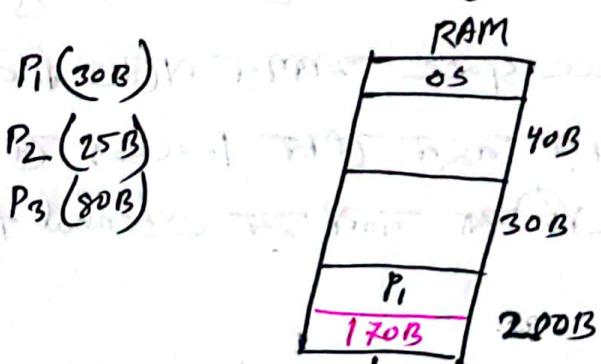
F_{DP} - P₂ (25 B)

$$50 - 25 = 25$$

Worst-Fit algorithm

ଏକାନ ପ୍ରକାର ହୋଲ୍ ଓ ପ୍ରକାର ଫଟିଗ୍ର ଅଲୋକେଶନ ଦ୍ୱାରା ଉତ୍ତରାଧିକାରୀ ହୁଏଥିବା ପ୍ରକାର

এবং allocation দ্বারা পুর অবচেতনা largest hole create হয়,



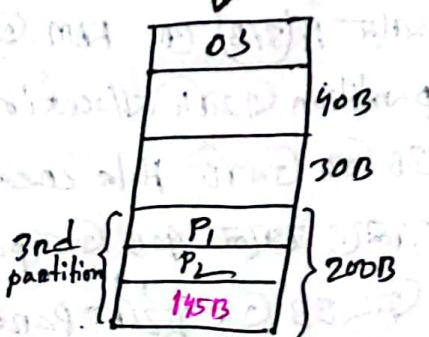
For P₁(30B)

$$\begin{aligned}40 - 30 &= 10 B \\30 - 30 &= 0 B \\200 - 30 &= 170 B\end{aligned}$$

For $P_2(25\beta)$

$$40 - 25 = 15B$$
$$30 - 25 = 5B$$

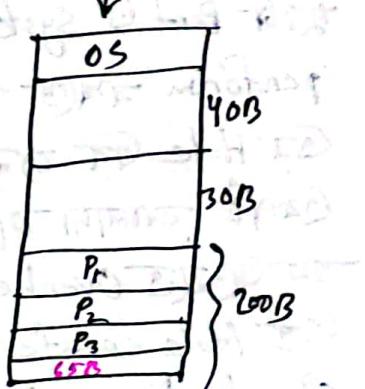
$$170 - 25 = 145B$$



For P₃ (80 B)

~~90~~ → X

$$145 - 80 = 65B$$

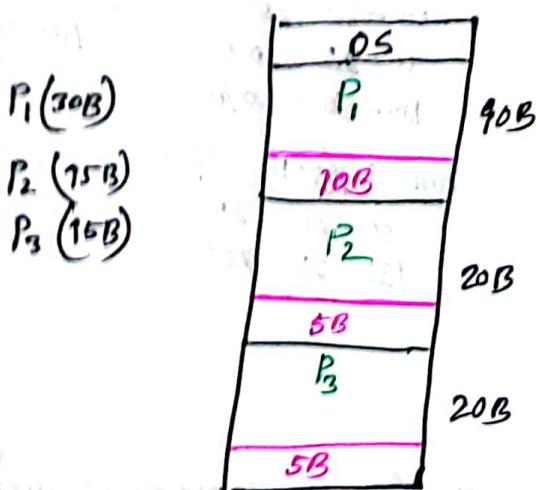


Fragmentation

Process (র) main memory (র) contiguous allocation (ক্ষেত্র সমষ্টি)

Fragmentation create (২), Fragmentation (ক্ষেত্র সমষ্টি)

External Fragmentation



$$\text{Total Hole size} = 10 + 5 + 5 = 20 \text{B}$$

(ক্ষেত্র 20B ক্ষেত্র RAM (র) contiguous
হলো)

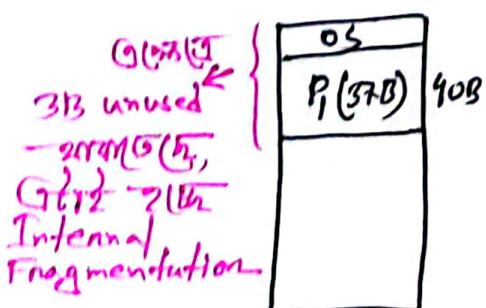
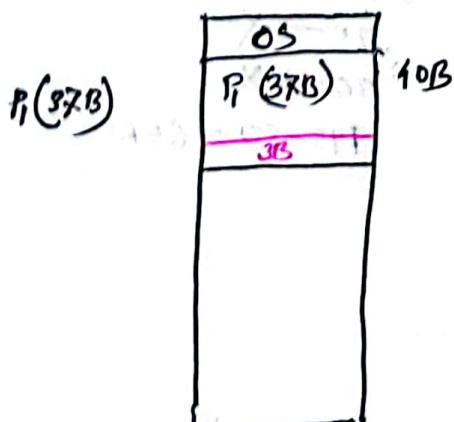
যেখানে প্রোসেস P₄ (15B) - ৩০B (র)
- ৩৫B (র) ক্ষেত্র Allocation (ক্ষেত্র সমষ্টি)
নি, যদিও Hole 20B > 15B,
বাকী 15B (র) contiguous space
হলো (ক্ষেত্র ক্ষেত্র বলো ২৫)
ক্ষেত্র External Fragmentation.

RAM (র) enough - এইসব Free space - মানব মাঝে (free space)

যদিও contiguous নি আবশ্যিক নয় তবে process (র) allocation

ক্ষেত্র ক্ষেত্র নি, (ক্ষেত্র issue ক্ষেত্র) ক্ষেত্র বলো ২৫ external fragmentation.

Internal Fragmentation



যেখানে P₁ (37B), (ক্ষেত্র RAM (র) 40B ক্ষেত্র ক্ষেত্র
partition (ক্ষেত্র ক্ষেত্র allocation ক্ষেত্র ক্ষেত্র ক্ষেত্র
3B ক্ষেত্র ক্ষেত্র Hole create - ক্ষেত্র physically
যান্তে ক্ষেত্র 40B ক্ষেত্র partition ক্ষেত্র 37B
(ক্ষেত্র 3B ক্ষেত্র ক্ষেত্র Part create - ক্ষেত্র ক্ষেত্র
ক্ষেত্র Part (ক্ষেত্র System (ক্ষেত্র different operation
perform ক্ষেত্র - ক্ষেত্র ক্ষেত্র 3B
ক্ষেত্র Hole ক্ষেত্র ক্ষেত্র ক্ষেত্র system (ক্ষেত্র
ক্ষেত্র ক্ষেত্র ক্ষেত্র ক্ষেত্র ক্ষেত্র perform ক্ষেত্র
- ক্ষেত্র ক্ষেত্র Overhead ক্ষেত্র ক্ষেত্র ক্ষেত্র ক্ষেত্র
(ক্ষেত্র Hole create ক্ষেত্র ক্ষেত্র ক্ষেত্র ৩০B ক্ষেত্র
allocation process P₁ (37) ক্ষেত্র ক্ষেত্র ক্ষেত্র ক্ষেত্র

* Negligible part or unused space after allocation এবং শৈলৰ কথা
বিধা অন্তর্ভুক্ত হওয়া পাই (internal fragmentation বোকান্তে),

* First-fit algorithm বোকান্তে Statistical analysis আওয়া
জাইসে, যদি N bit blocks RAM এ allocate কৰা হ'ব তবে
 $0.5 \times N$ blocks lost of fragmentation. $\frac{1}{2}$ statistic অনুমান
RAM কৰ $\frac{1}{3}$ portion unusable থাক, which is known as
50-percent rule.

* System কৰ Internal fragmentation বোকান্তে Remove কৰা কাম না
কৰ্তৃত external fragmentation বোকান্তে Remove কৰা কাম,

* External fragmentation remove কৰা কাম - কোটি elementary
approach - Compaction.

* Compaction কৰ memory কৰ content কুলেক্ট পুনরাবৃত্ত সুফল
কৰা হ'ব এবং কুল কুল অন্তর্ভুক্ত allocated memory space কুল
কোভার কৰা হ'ব Hole কুলে কোভার কৰা হ'ব বোকান্তে
Hole কৰ large কোভি block create কৰ এবং পুনরাবৃত্ত কৰিব
size কৰ Process allocate কৰা হ'ব পাই,

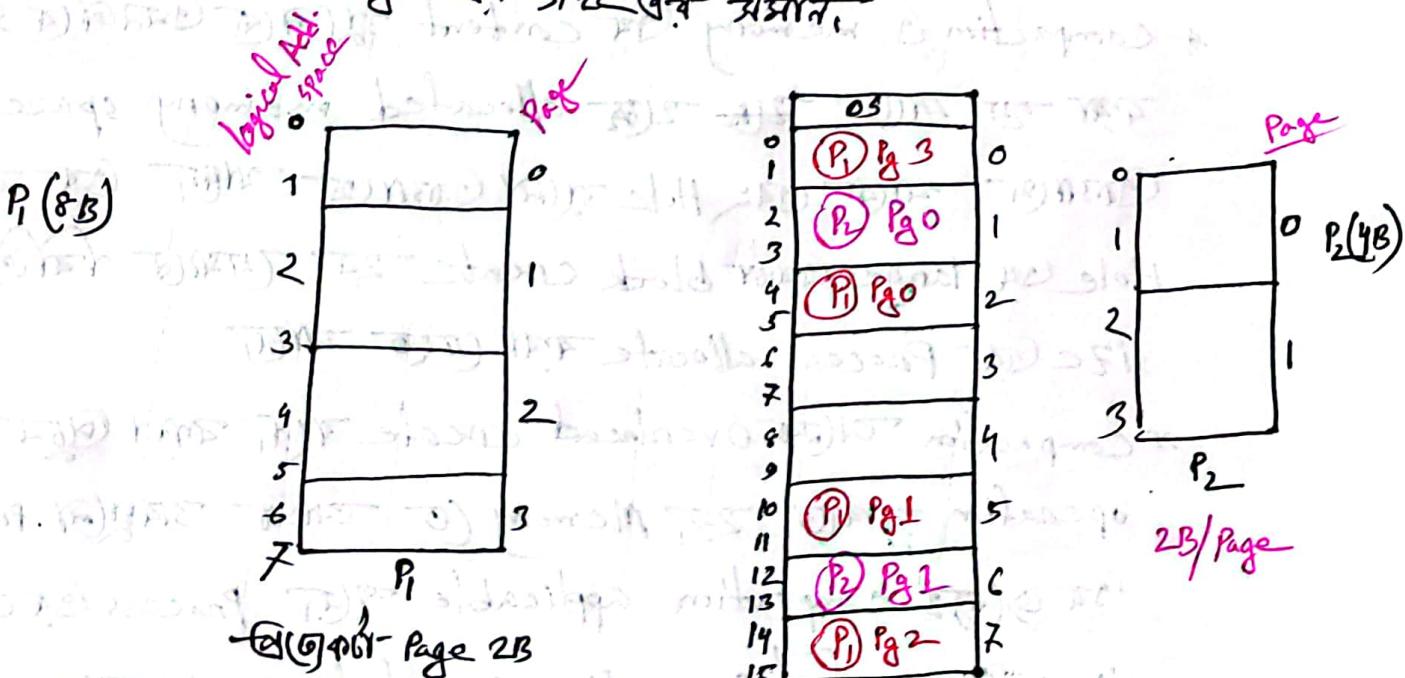
* Compaction অন্তর্ভুক্ত Overhead create কৰা, কোভি কুল Complex
operation কৰা হ'ব, Memory কৰ কোভি - অবগুলি process
কৰ কোভি compaction applicable হ'ব, process কৰ execution
time কৰ কোভি Compaction complete কৰা হ'ব

Paging

Paging এর main purpose of ~~to~~ external fragmentation to efficient way to remove ~~RAM~~

Modern OS গ process টি RAM গ allocation করেছোৱ জন্য Paging use কৰিব হৈব।

Paging গত through কোৱ process contiguously RAM গ allocation কৰিব।
কোৱ process এর logical Address Space আলগোকো same size এবং multiple part কো divide কৰা হৈব। এবং multiple part কো divide কৰা হৈলে, একটি অিত্রুণিৰ্দিত part কো বলা হৈব page। Page এর size স্থিৰ হৈব। RAM আলগ কো same size এবং multiple part কো divide কৰা হৈব। এভৰys RAM এর অিত্রুণিৰ্দিত part এর size কোটা Page এর size এবং অসমান।



অিত্রুণিৰ্দিত - Page 2B

পৰি, কোৱ মতুলে
process প্ৰাপ্তি কৰাৰ
অসূচি same হৈব।

RAM কো কো অিত্রুণিৰ্দিত
part কো divide কৰা হৈলে,
একটি অিত্রুণিৰ্দিত part কো
বলি হৈব frame.

process के लिए storage component (HDD, SSD) store करता है।
जिसकी गहराई same size के multiple page को divide करता है।



* Page size = frame size

Page size $\approx 2^n$ [2, 4, 8, 16, 32, ...]

Process के size 2^m के equivalent होते हैं। mandatory है।

* 1 frame के multiple page allocate कर सकता है।

* 2^m Process के Page जूले RAM के frame के allocation
द्वारा होते हैं। RAM के Process के अन्य कानून सेपाने
Page table store करता है।

P₁ (Page table)

#Page	#Frame
0	2
1	5
2	7
3	0

P₂ (Page table)

#Page	#Frame
0	1
1	6

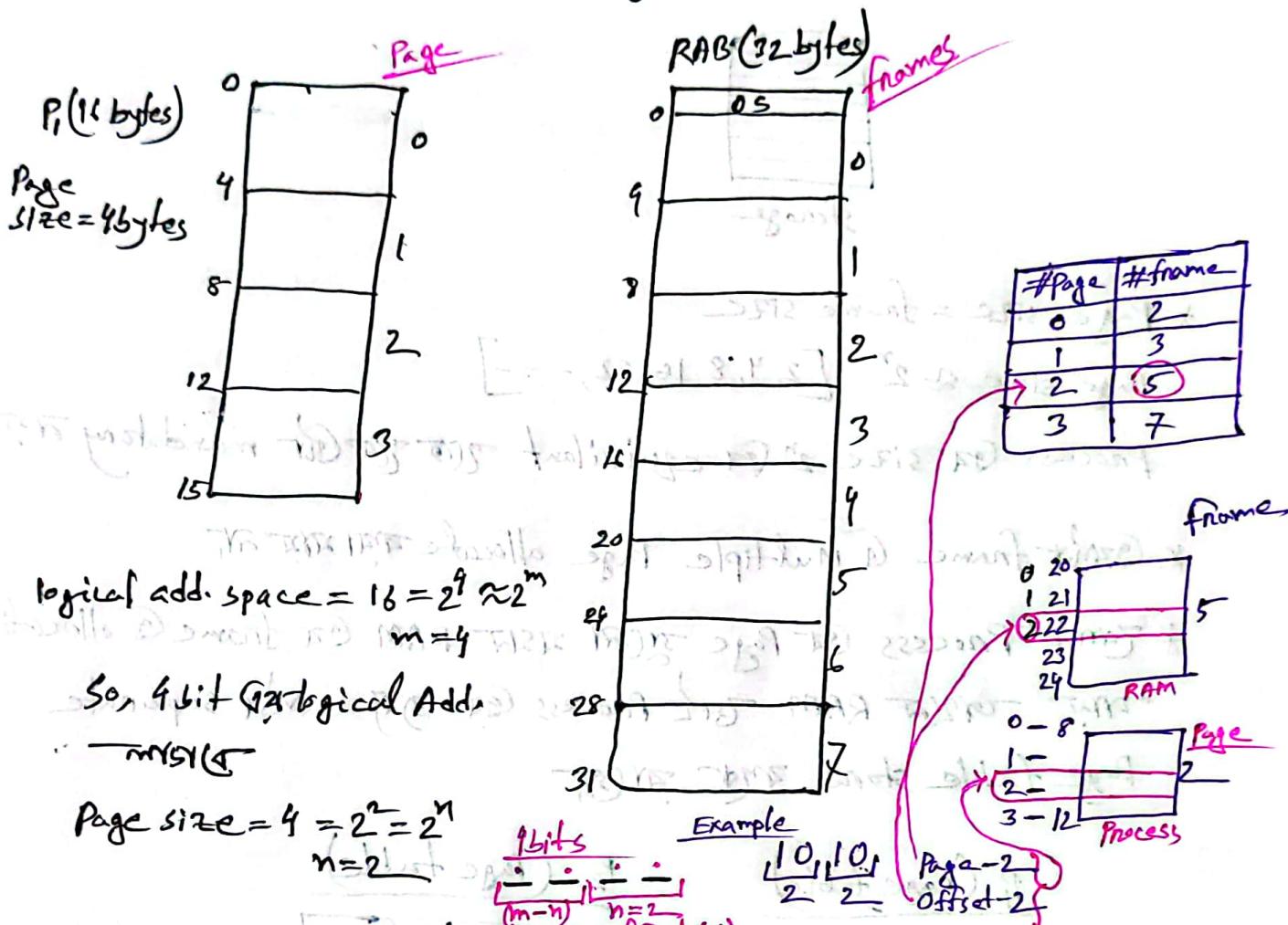
Page table logical Address का Physical Address को translate करता है।

* different size के memory chunks का problem remove करता है। Paging use करता है।

* Paging apply करते internal fragmentation avoid कर सकता है।

#Address Translation Schema

Address translation system uses Page table we have 2⁴



CPUs generate instructions execute them to generate Logical Address, generate page number and logical address bit to get corresponding physical address space. Instructions like load, store, etc.

Process size 16 bytes \rightarrow 4 pages of 4 bytes logical address space to represent $2^4 \rightarrow 4$ bit address.

Another example

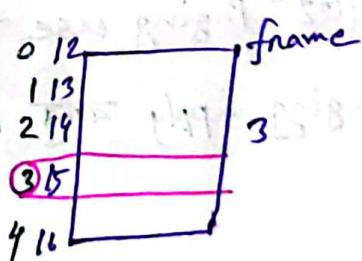
$\frac{0111}{(m-n)}$

Page $\rightarrow 1$
Offset $\rightarrow 3$

Page table

For example
Given -

#Page	#frame
1	3

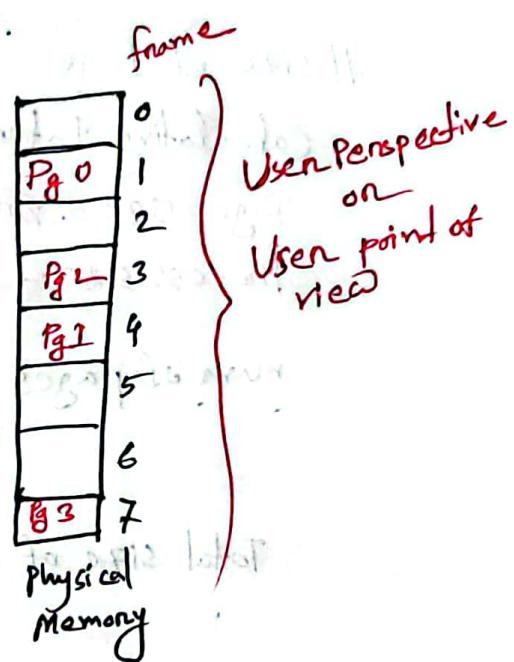


Paging Model of Logical and Physical Memory.

Page 0
Page 1
Page 2
Page 3

Page	frame
0	1
1	4
2	3
3	7

Page table



Paging Example

Given, $n=2$ & $m=4$

32 bytes memory

$$\text{Page size} = 2^n = 2^2 = 4$$

$$\text{Logical Address Space} = 2^m = 2^4 = 16$$

memory size 32 bytes base 2⁴ bytes
 value 2⁴ bytes 2²,
 16 bytes 32 bytes $\rightarrow 2^m$
 $\therefore m=5$

Page	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	a															
1	b															
2	c															
3	d															
4	e															
5	f															
6	g															
7	h															
8	i															
9	j															
10	k															
11	l															
12	m															
13	n															
14	o															
15	p															

Logical Memory

Page	frames
0	5
1	6
2	1
3	2

logical Add: $\rightarrow 2$

$2 \rightarrow 0010_2$
 $(m-n)$ offset

page num
 \downarrow
 0

0	-															
9	-															
8	-															
7	-															
12	-															
16	-															
0	20	-														
1	21	-														
2	22	-														
3	23	-														
24	-															
25	-															
26	-															
27	-															
28	-															
29	-															
30	-															
31	-															

logical Add $\rightarrow 2$
 $2 \rightarrow 00010_2$
 $(m-n)$ offset
 page num
 0

Size of Page.

Calculating Internal fragmentation

Page size = 2048 bytes

Process size = 72766 bytes

$$\text{num of pages} = \frac{72766}{2048}$$
$$= 35.53 \approx 36 \text{ pages}$$

$$\text{Total size of Pages} = 36 \times 2048$$
$$= 73728 \text{ bytes}$$

~~more Required size~~ = 73728

$$\text{Internal fragmentation} = 73728 - 72766$$
$$= 962 \text{ bytes}$$

Worst Case

1 frame - 1 byte internal fragmentation

$$\text{Fragmentation} = 1 \text{ frame} - 1 \text{ bytes}$$

Average Case

1 frame - $\frac{1}{2}$ frame internal fragmentation

$$\text{Fragmentation} = \frac{1}{2} \times \text{frame size}$$

Page size upto 2¹⁶ bytes internal fragmentation

— কম সূচী বিহুর প্রয়োজন অসুবিধা inefficient Garbage mechanism,

— বাধা প্রেক্ষিতে Page-table দ্বি- entry আভ্যন্তরীণ মান,

— OSW Page table দ্বি- entry RAM এ জোড়া space

— LSAT Page table দ্বি- entry RAM এ জোড়া information lookup complexity অসুবিধা

- * Solaris OS ৰ Page Size use বৰ্তমান ৰ ৱৰ্তন \rightarrow 8KB and 4MB
 - * Paging কৰা ব্যাবটো Process এবং গ্ৰেটৰ View দিয়ে Physical memory
এবং different গ্ৰেটৰ View আছে যাইকৈ,
 - * Process physical memory কৰে access কৰিবলৈ, Process
পুনৰুৎপন্ন গ্ৰেটৰ memory কৰে access কৰিবলৈ
 - * ক্ষেত্ৰ ক্ষেত্ৰ free অৱস্থা অৰ্থাৎ ক্ষেত্ৰ ক্ষেত্ৰ frame
ও process allocate কৰিবলৈ মৈত্ৰি System কৰে ক্ষেত্ৰ আছে,

Implementation of Page Table

- *- लिटरेन्ट प्रोसेस द्वारा सेपारेट पेज टेबल मैनीमी
स्टोक एवं डाक्ट्रो

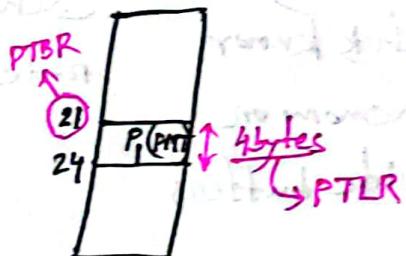
- * Page table implementation in RAM (গুরুত্বপূর্ণ Register use)

- Page-table base Register (PTBR)

RAM એ કોઈ particular address space એવી page table
એ એ એ allocation એવી એવી indicate કરું એ PTBR એ એ

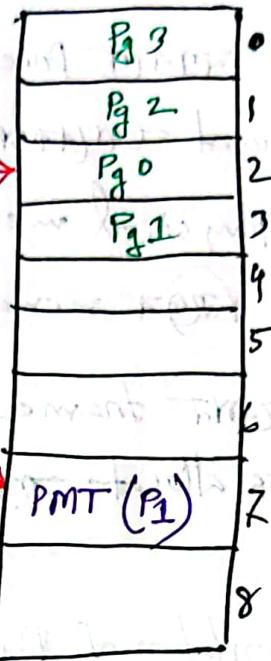
• Page Table Length Register (PTLR)

Q) If use 8MB page table, what size indicate
ব্যাপ্তি



Page Table (PMT) - P_i

#Page	#frame
0	2
1	3
2	1
3	0



जैसे कि particular

page द्वारा instructions को

execute करना होता है CPU

इसका main memory का

access जल्दी होता है किंतु

Page table finding होता है

प्रोцेसर के लिए Page

table जो कि page info

अनुसारी Particular frame

का access होता है,

कि particular Page द्वारा instructions

execute होता है CPU को लिया

इसका main memory का

access जल्दी होता है किंतु

ऐसी ही cache memory

होती है, which is known

as associative memory or

translation look-aside buffers

(TLBs).

Page table का copy करना है Associative memory /

TLBs G. — इसने CPU Page table को first G. main memory

to — Look up करता है, TLB का cache memory G. look up करता



मिले CPU G. main memory का

ऐसी ही access जल्दी 200 ms अभी

लागे होते हैं कि Page द्वारा

instruction execute करना है

CPU G. main memory का access

किते total 900 ms time लगता

है और overhead create करता

है सभी — अधिकारी बनते हैं

use करते हैं तो G. cache —

Cache memory.

Cache memory G. data transfer

rate RAM का तुलना

Cache memory জন্য required frame Gt information
 CPU main memory Gt particular frame Gt
 access করেছে, একটি main memory Gt CPU Gt পেজ অ্যাসিস্টেন্স
 Lookup করা - প্রোচেসর দ্বারা

Associative Memory

Page	frame
0	3
1	4
2	1
3	5

Let's say,

CPU generated logical Address
 to page num 1

Now parallel search on Associative
 memory's page table for page 1

Now CPU will take access on
 the particular frame for page 1
 of main memory.

Gt New Page number

CPU generated Gt

Gt Associative

memory Gt new page

no Gt info আপডেট না, Gt

CPU main memory Gt lookup

বল্টে Page 4 এর জন্য, Gt

main memory Gt store করা

Page table Gt করা new

page 4 খুঁড়ে আনা Gt করা

information Gt associative
 memory Gt update করা

একেতে CPU করা 3 বার করা

বল্টে ৩টি, Gt associative

memory Gt করা হয়েছে main

memory Gt

TLB Gt Page table G access

বল্টে মনিপুলেট পার্টিকুলার page

Gt info. আপডেট করা Gt

বল্টে TLB hit. - Gt করা

info আপডেট করা যায়, একে বল্ট

TLB miss.

TLB hit

Look up \rightarrow 1 Ass. M. + 1 RAM

TLB Miss

Look up \rightarrow 1 Ass. M. + 2 RAM

Effective Access Time

Logical Add. \rightarrow Physical Address \rightarrow translate \rightarrow total
addr-unit time spent \rightarrow $\frac{1}{2}$ of $\frac{1}{2}$ effective access
time

$E \rightarrow$ Associative lookup time

$\alpha \rightarrow$ Get associative memory/TLB (\rightarrow hit ratio)

Associative memory (\rightarrow lookup \rightarrow total \rightarrow $\frac{1}{2}$)

particular page (\rightarrow corresponding frame
 \rightarrow info off \rightarrow $\frac{1}{2}$, $\frac{1}{2}$ \rightarrow $\frac{1}{2}$ Hit ratio.

TLB Hit (\rightarrow $\frac{1}{2}$ Percentage \rightarrow $\frac{1}{2}$ Hit ratio)

Given,

$$\alpha = 80\% = 0.8 \quad \text{RAM Access time} = 100 \text{ ns}$$

$$E = 20 \text{ ns} \quad (\text{using } 50\%)$$

$$\text{Miss ratio} = 1 - 0.8 = 0.2$$

TLB Hit \rightarrow $\frac{1}{2}$ time spent \rightarrow $\frac{1}{2}$ TLB miss \rightarrow
 \rightarrow time spent \rightarrow $\frac{1}{2}$ (\rightarrow total time \rightarrow Summation
 \rightarrow $\frac{1}{2}$ Effective Access Time (EAT)).

$$\text{EAT} = \text{TLB Hit time} + \text{TLB Miss time}$$

$$= [0.8 \times (20 + 100)] + [0.2 \times (20 + (2 \times 100))]$$

$$= 140 \text{ ns}$$

$$\frac{\text{TLB Hit}}{1 \text{ Ass M} + 1 \text{ RAM Look up}}$$

$$\frac{\text{TLB Miss}}{1 \text{ Ass M} + 2 \text{ RAM Look up}}$$

~~#EAT~~ problem

$$\infty = 99\% = 0.99$$

$$E = 20 \text{ ns}$$

$$\text{Miss ratio} = 1 - 0.99 \\ = 0.01,$$

RAM access time = 100 ns

$$EAT = TLB \text{ Hit time} + TLB \text{ miss time}$$

$$= [0.99 \times (20 + 100)] + [0.01 \times \{20 + (2 \times 100)\}]$$

= 121 ns

Shared Pages

* Shared code

process ଏକ code ଦ୍ୱାରା ଏମନ୍ତଙ୍କ section ଯେଉଁ section ଦ୍ୱାରା
code for multiple process ଦ୍ୱାରା ଅଣ୍ଟିଷ୍ଟିକ୍ଷଣ କରାଯାଇଥାଏ

Read only mode (a).

- Shared code GA concept for Inter Process Communication
GA IS NOT effective
 - Similar to multiple threads sharing the same process space.

* private code and data

• Copyright process or कानूनी कोड ग्र- individual copy

ପ୍ରକାଶ ପ୍ରତ୍ୟୋଗିତା ଅନୁକ୍ରମରେ Process ଏବଂ ଆମେ Share କରିବାକୁ

କ୍ଷେତ୍ର ପାଇଁ କ୍ଷେତ୍ର ପାଇଁ Process କିମ୍ବା private code କିମ୍ବା data

Page ৩০৫ Process ট্র্যাক-
Private code/data - মন্তব্য করুন

Logical address space କେବଳ ପ୍ରକାଶିତ address ଓ ଅନ୍ତରାଳ ମଧ୍ୟରେ

* Paging କେନ୍ଦ୍ରିୟରେ କାରତେ ଏହି ଧ୍ୟାନ ମୁଣ୍ଡରେ Page G shared code

— ଅମ୍ବାତ୍ମକ ପାଇଁ Page 6 ଜ୍ୟୋତିଷ ପ୍ରିସ୍ କୋଡ୍ ଏବଂ ଡାଟା ଅଧ୍ୟକ୍ଷଣ

-女兒在 Page 1 private code and data - name 751

Page 9 ~~and~~ Shared Code ~~and~~ Mixed ~~and~~

RAM ৰে প্রিগ্ৰহণ

Process ৰে shared
code কৰা page কৰলো

memory allocation কৰিব

-Or অনেক নিচেভুলি

শেষ দাটা মেমৰি বাধা

Shared code ৰে
process ৰে same

প্রিগ্ৰহণ

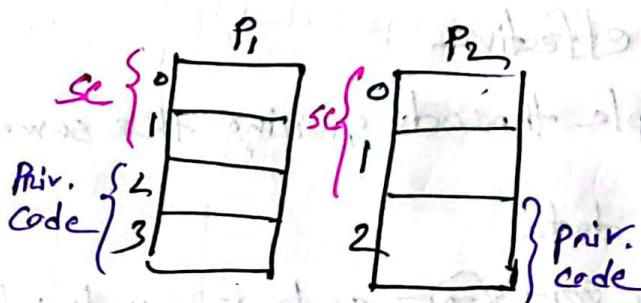
Q: Problem remove

বাবে শেষ page ৰে

concept introduce কৰ

প্ৰিগ্ৰহণ

-প্রিগ্ৰহণ process ৰে shared code কৰা page কৰলো - same
order ৰে বাধা কৰিব



RAM ৰে এই কৰা process ৰে
জন্য ০ কৰা ১ no page
বেঁচাবলৈ allocate কৰা হৈ

Shared Page Example

P ₁	Pg	frame
cd1	0	3
cd2	1	9
cd3	2	6
data1	3	1

P ₂	Pg	frame
cd1	0	3
cd2	1	4
cd3	2	6
data2	3	7

P ₃	Pg	frame
cd1	0	3
cd2	1	1
cd3	2	6
data3	3	2

data1	1
data3	2
cd1	3
cd2	4
cd3	5
data2	6
cd1	7
cd2	8
cd3	9
data1	10
data3	11

Question

In a system, there are two processes - P₁(16 bytes), P₂(8 bytes) with page size of 4 bytes. Size of the main memory 32 bytes.

Page tables of processes are given below

# Page	# frame
0	3
1	7
2	10
3	5

# Page	# frame
0	4
1	0

PMT of P₁

Find corresponding Physical Address of the following logical addresses.

- a) 10111 of P₁
- b) 00101 of P₁
- c) 11011 of P₂

- d) 01010 of P₂
- e) 1011 of P₁
- f) 1111 of P₂

$P_1 \rightarrow 16 \text{ bytes} \rightarrow 2^4 \rightarrow 4 \text{ bit}$ represent 16 bit logical Address

space to represent $\therefore m = 4$

$P_2 \rightarrow 8 \text{ bytes} \rightarrow 2^3 \rightarrow 3 \text{ bit} \therefore m = 3$

Page size $\rightarrow 4 \text{ bytes}$ $2^2 \Rightarrow n = 2$ \therefore Right side represent 2 bit
bits offset represent

RAM size $\rightarrow 32 \text{ bytes} = 2^5 \rightarrow m = 5$ - \therefore $m = 5$

a) 10111

$m = 5$

$\begin{array}{r} 10111 \\ \hline (m-n) \quad n \\ (5-2) \quad = 2 \\ = 3 \quad \text{offset} \\ \text{Page no.} \end{array}$

Page no $= 101 = 5$ [Invalid page num]

Offset $= 11 = 3$

b)

00101

$\begin{array}{r} 00101 \\ \hline 1 \quad 1 \end{array}$

Page $\rightarrow 1 \rightarrow$ frame $\rightarrow 7$ [from Page table]

Offset $\rightarrow 1$

Physical Address = {Frame num \times Pagesize} + Offset

$$= (7 \times 4) + 1$$

= 29 [Valid Physical address cause $(0-31)$]

c) 11011

$\begin{array}{r} 11011 \\ \hline 6 \quad 3 \end{array}$

Page $\rightarrow 6$ [Invalid Page]

Offset $\rightarrow 3$

d) 01010 of P₂

01010
2 2

Page → 2 [Invalid page]

offset → 2

e) 1011 of P₁

m = 4

m + n = 4 + 2 = 6

10111
2 3

Page → 2 → frame → 10

offset → 3

$$\begin{aligned}\text{Physical Address} &= (10 \times 4) + 3 \\ &= 43 \text{ [Invalid Physical Address]}\end{aligned}$$

f) 1111 of P₂

11111
3 3

Page → 3 [Invalid Page no]

Offset → 3

- {
- RAM size 40 bytes $\rightarrow 2^6 \text{ bits} \rightarrow 6 \text{ bit}$
 - 101000 $\rightarrow 6 \text{ bit}$
So, m = 6
 - Page size 3 bytes $\rightarrow 2^2 \text{ bits} \rightarrow 2 \text{ bits}$
 - 11 $\rightarrow 2 \text{ bits}$
So, n = 2

Question

Assume that, Page size = 8 bytes and Physical memory = 64 bytes.
 If the CPU generates logical addresses 16, 6, 77, 21, 14, 3, 19 and 5, respectively then how can the user's view of memory be mapped into Physical memory?

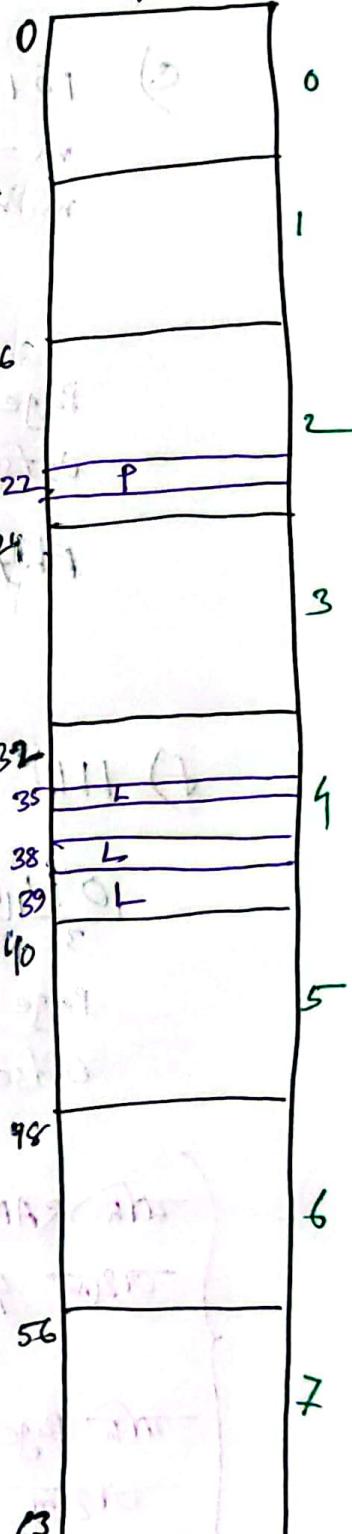
Logical Memory

Page #	Data
P ₀	P
P ₁	P
P ₂	C
P ₃	S
P ₄	T
P ₅	E

Page #	Frame #
P ₀	4
P ₁	2
P ₂	10
P ₃	7
P ₄	9
P ₅	0

PMT

RAM



$$\text{Page Size} = 8 = 2^3 \Rightarrow n = 3$$

$$\text{RAM Size} = 64 = 2^6 \Rightarrow m = 6$$

or

$$\text{Total Process size} \rightarrow 6 \times 8$$

= 48 bytes

$$48 \rightarrow 110000 \rightarrow 6 \text{ bits}$$

$$m = 6$$

$$m - n = 6 - 3 = 3$$

$$\text{Num of frame} = \frac{64}{8} = 8$$

$$\text{Physical Add} = \{\text{Frame num} \times \text{Page size}\} + \text{offset}$$

$$\textcircled{1} \quad 16 \rightarrow \begin{array}{l} \underline{010\ 000} \\ \text{Page no} \quad \text{Offset} \\ \hline 2 \quad 0 \end{array}$$

Page \rightarrow 2, frame \rightarrow 10

offset \rightarrow 0

$$PA = (10 \times 8) + 0 = 80 \quad [\text{Invalid PA}]$$

User View of Memory

$$② 6 \rightarrow \underline{\underline{000110}}$$

page $\rightarrow 0$ frame $\rightarrow 4$

offset $\rightarrow 6$

$$\begin{aligned} PA &= (4 \times 8) + 6 \\ &= 38 \end{aligned}$$

$$③ 19 \rightarrow \underline{\underline{010011}}$$

Page $\rightarrow 2$ frame $\rightarrow 10$

Offset $\rightarrow 3$

$$\begin{aligned} PA &= (10 \times 8) + 3 \\ &= 83 \quad [\text{Invalid PA}] \end{aligned}$$

④ 77 \rightarrow invalid logical Add.
6 bit represent
not possible

$$⑤ 21 \rightarrow \underline{\underline{010101}}$$

Page $\rightarrow 2$, frame $\rightarrow 10$

offset $\rightarrow 5$

$$\begin{aligned} PA &= (10 \times 8) + 5 \\ &= 85 \quad [\text{invalid PA}] \end{aligned}$$

$$⑥ 5 \rightarrow \underline{\underline{000101}}$$

Page $\rightarrow 0$ frame $\rightarrow 4$

Offset $\rightarrow 5$

$$\begin{aligned} PA &= (4 \times 8) + 5 \\ &= 39 \end{aligned}$$

$$⑦ 14 \rightarrow \underline{\underline{001110}}$$

Page $\rightarrow 1$ frame $\rightarrow 2$

Offset $\rightarrow 6$

$$\begin{aligned} PA &= (2 \times 8) + 6 \\ &= 22 \end{aligned}$$

$$⑧ 3 \rightarrow \underline{\underline{000011}}$$

Page $\rightarrow 0$ frame $\rightarrow 4$

Offset $\rightarrow 3$

$$\begin{aligned} PA &= (4 \times 8) + 3 \\ &= 35 \end{aligned}$$