Branch Prediction

Nathan Beckmann

CMU

Based on slides by Joel Emer, MIT

Phases of Instruction Execution

PC

I-cache

Fetch Buffer

Issue Buffer

Func. Units

Result Buffer

Arch. State

Fetch: Instruction bits retrieved from cache.

Decode: Instructions placed in appropriate
ssue/reorder/storage buffer

Execute: Instructions and operands sent to execution units .
When execution completes, all results and exception flags are available.

Commit: Instruction irrevocably updates architectural state (aka "graduation" or "completion").

Beckmann

Control Flow Penalty

Next fetch
started

*Modern processors may have
> 10 pipeline stages between
next PC calculation and branch
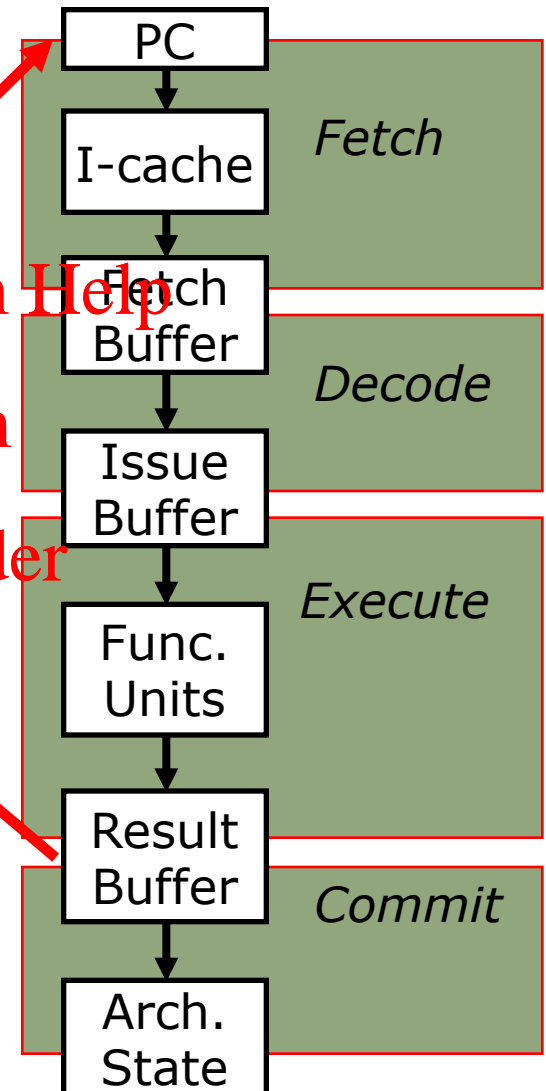resolution !*

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

Loose loop

*How much work is lost if
pipeline doesn't follow
correct instruction flow?*

Branch
executed

~ Loop length x pipeline width

```
PC
 ↓
I-cache        Fetch
 ↓
Fetch
Buffer
               Decode
 ↓
Issue
Buffer
               Execute
 ↓
Func.
Units
 ↓
Result
Buffer         Commit
 ↓
Arch.
State
```

Beckmann

Average Run-Length between Branches

Average dynamic instruction mix from SPEC92:

	SPECint92	SPECfp92
ALU	39 %	13 %
FPU Add		21 %
FPU Mult		13 %
load	26 %	23 %
store	9 %	9 %
branch	16 %	8 %
other	10 %	12 %

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

SPECint92:   *compress, eqntott, espresso, gcc , li*
SPECfp92:   *doduc, ear, hydro2d, mdijdp2, su2cor*

What is the average *run length* between branches

Beckmann

MIPS Branches and Jumps

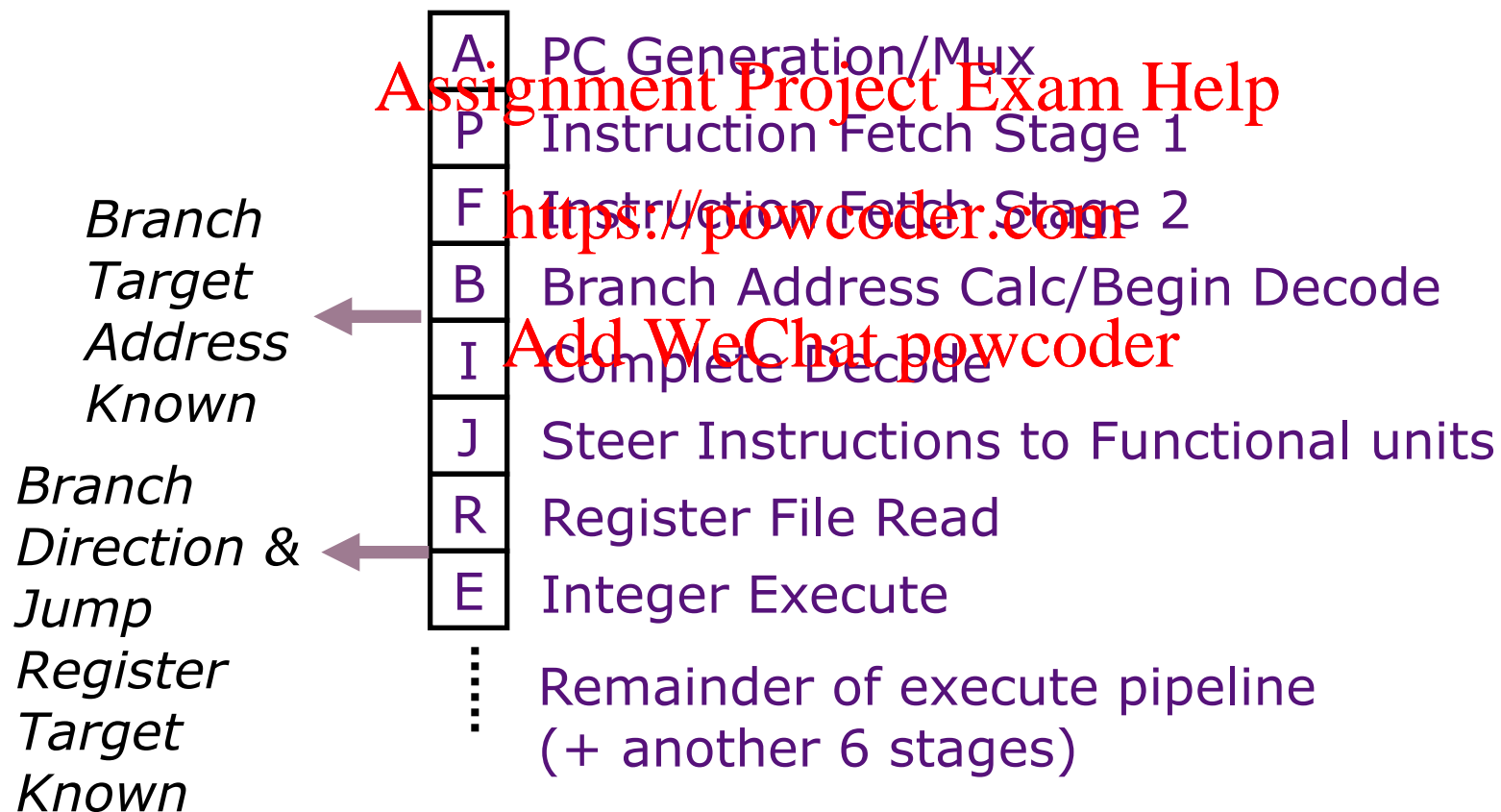Each instruction fetch depends on one or two pieces of information from the preceding instruction:

1) Is the preceding instruction a **taken branch**?

2) If so, what is the target address?

Instruction	Taken known?	Target known?
J	After Inst. Decode	After Inst. Decode
JR	After Inst. Decode	After Reg. Fetch
BEQZ/BNEZ	After Reg. Fetch[*]	After Inst. Decode

[*]Assuming zero detect on register read

Beckmann

Realistic Branch Penalties

UltraSPARC-III instruction fetch pipeline stages
(in-order issue, 4-way superscalar, 750MHz, 2000)

A.	PC Generation/Mux
P	Instruction Fetch Stage 1
F	Instruction Fetch Stage 2
B	Branch Address Calc/Begin Decode
I	Complete Decode
J	Steer Instructions to Functional units
R	Register File Read
E	Integer Execute
	Remainder of execute pipeline (+ another 6 stages)

Branch Target Address Known

Branch Direction & Jump Register Target Known

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

Beckmann

Reducing Control Flow Penalty

Software solutions

- *Eliminate branches - loop unrolling*

  Increases the run length

- *Reduce resolution time - instruction scheduling*

  Compute the branch condition as early

  as possible (of limited value – why?)

Hardware solutions

- Find something else to do architecturally

  - *delay slots* - replace pipeline bubbles with useful work (requires software cooperation)

- *Speculate - branch prediction*

  Speculative execution of instructions beyond the branch

  Useful in both in- and out-of-order processors

Branch Prediction

Motivation:

Branch penalties limit performance of deeply pipelined processors

Modern branch predictors have high accuracy (>95%) and can reduce branch penalties significantly

Required hardware support:

Prediction structures:
- Branch history tables, branch target buffers, etc.
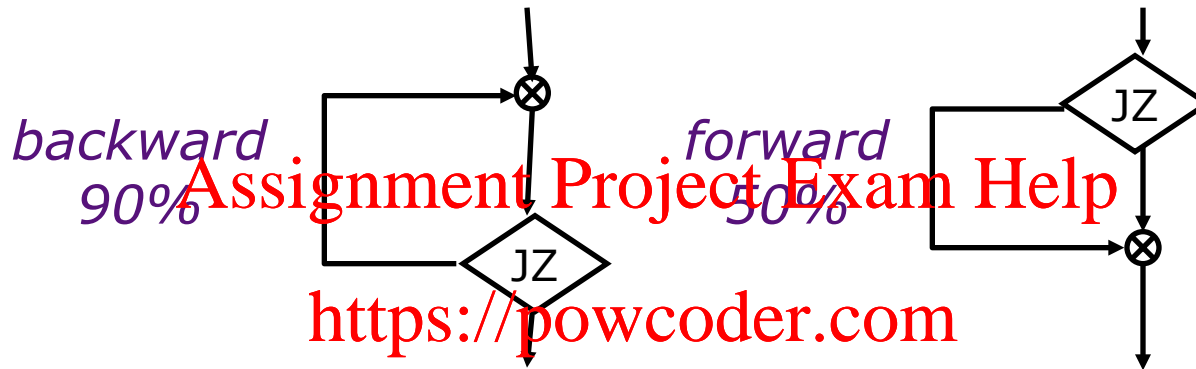
Mispredict recovery mechanisms:
- *Keep result computation separate from commit*
- Kill instructions following branch in pipeline
- Restore state to state following branch

Beckmann

Static Branch Prediction

Overall probability a branch is taken is ~60-70% but:

backward
90%

forward
50%

JZ

JZ

ISA can attach preferred direction semantics to branches, e.g., Motorola MC88110

   bne0 *(preferred taken)*   beq0 *(not taken)*

ISA can allow arbitrary choice of statically predicted direction, e.g., HP PA-RISC, Intel IA-64

    typically reported as ~80% accurate

Beckmann

Is Static Prediction Enough?

- UltraSPARC-III
  - Mispredicted branches have penalty of 6 cycles
  - 4-wide issue

- Wasted work per branch @ 80% accuracy:

20% misprediction rate
× 6 stages
× 4-way issue
= 4.8 wasted instructions / branch

- Branches are 15-20% of instructions!

Dynamic Branch Prediction
learning based on past behavior

Temporal correlation

The way a branch resolves may be a good predictor of the way it will resolve at the next execution
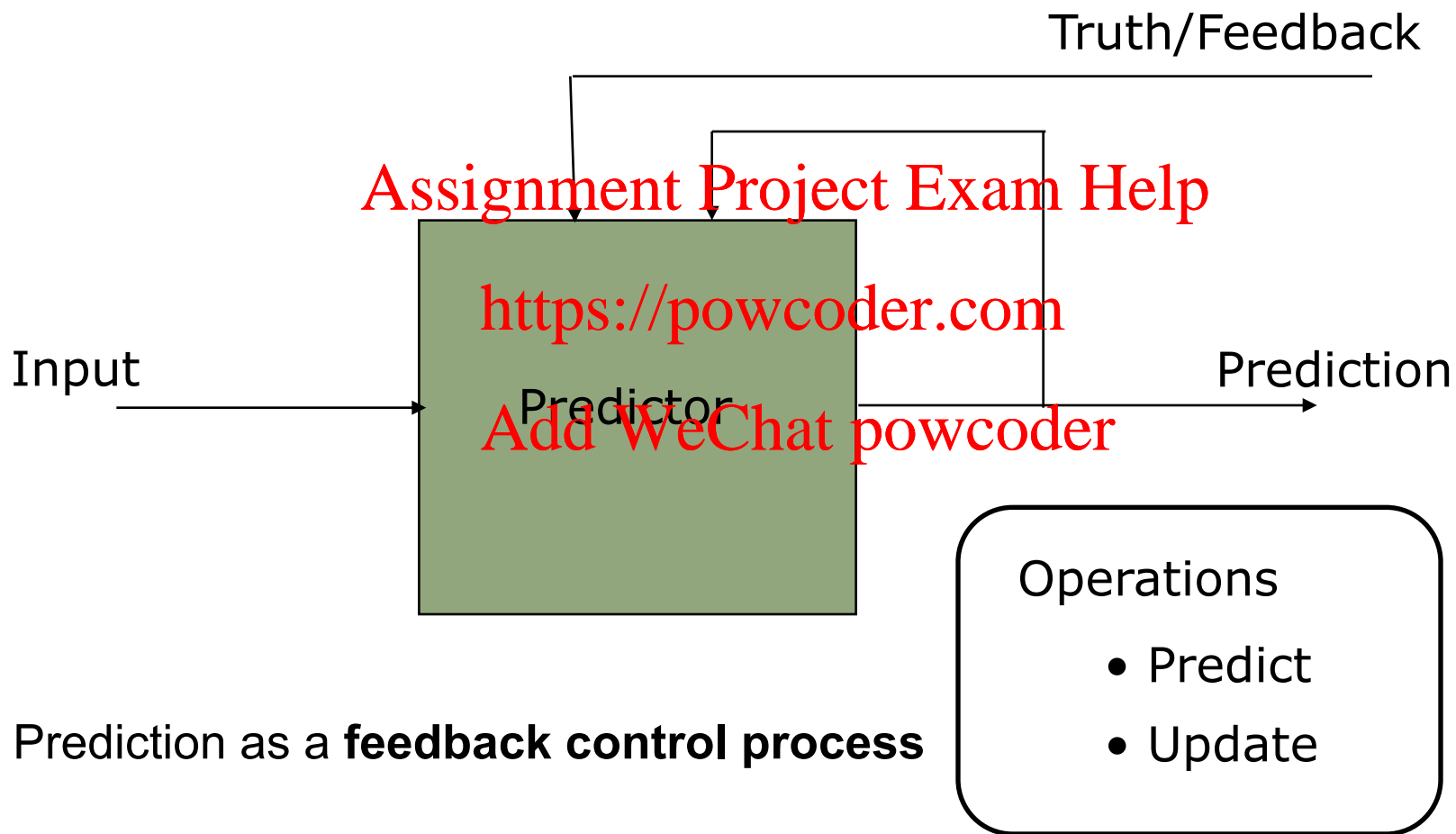
Assignment Project Exam Help

https://powcoder.com

Spatial correlation Add WeChat powcoder

Several branches may resolve in a highly correlated manner *(a preferred path of execution)*

Echoes of temporal/spatial locality in caches…

Beckmann

Dynamic Prediction

Truth/Feedback

Input

Predictor

Prediction

Operations

- Predict

- Update

Prediction as a **feedback control process**

Beckmann

Predictor Primitive
Emer & Gloy, 1997

- Indexed table holding values

- Operations
  - Predict
  - Update

Index

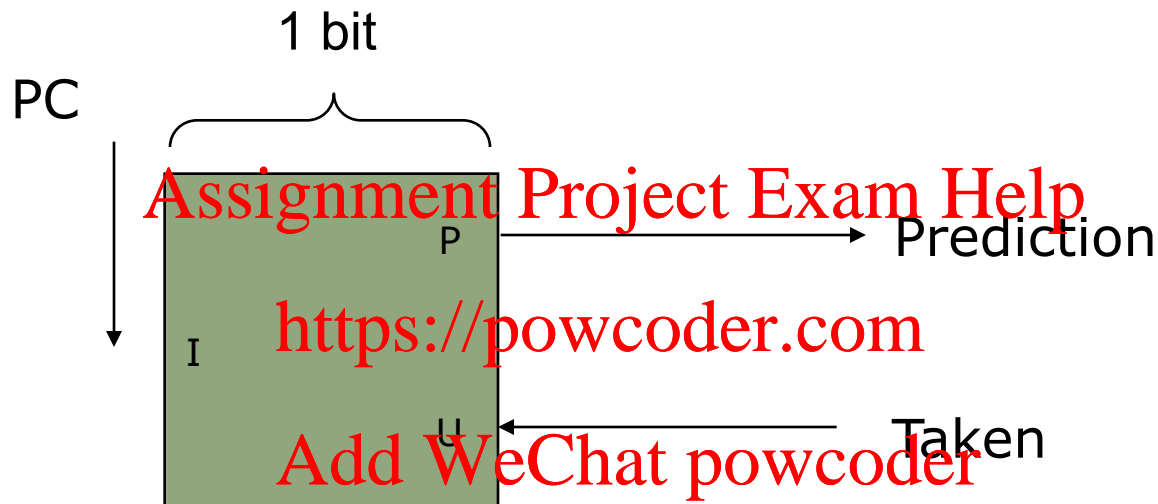Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

Depth

Width

P

I

U

Prediction

Update
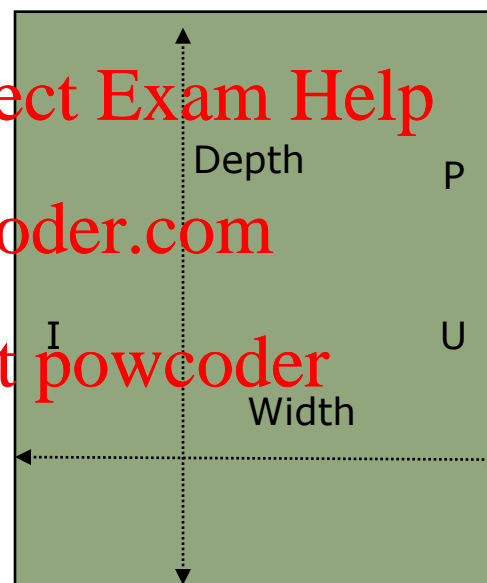
- Algebraic notation

Prediction = P[Width, Depth](Index; Update)

One-bit Predictor

Simple temporal prediction

1 bit

PC

P → Prediction

I

U ← Taken

A21064(PC; T) = P[1, 2K](PC; T)

What happens on loop branches?

At best, mispredicts twice for every use of loop.

Beckmann

Branch Prediction Bits

- Assume 2 BP bits per instruction
- Use saturating counter

On ¬taken	↑ On taken			
		1	1	Strongly taken
		1	0	Weakly taken
		0	1	Weakly ¬taken
		0	0	Strongly ¬taken

Beckmann

Two-bit Predictor
Smith, 1981

2 bits

PC

I

P

Prediction

Taken
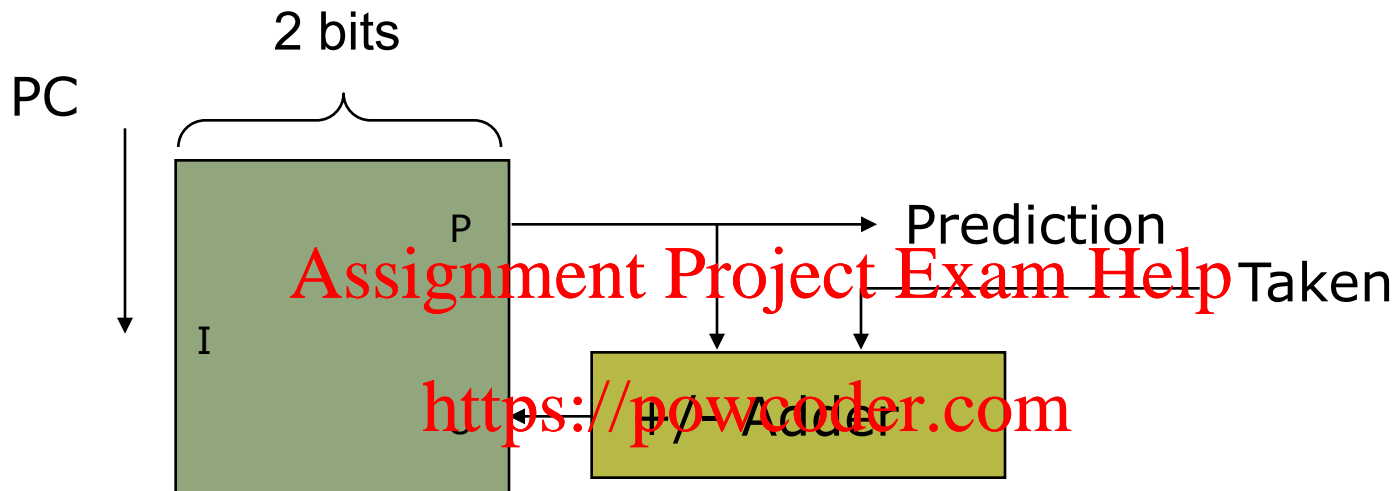
+/- Adder

Counter[W,D](I; T) = P[W, D](I; if T then P+1 else P-1)

A21164(PC; T) = MSB(Counter[2, 2K](PC; T))

27 April 2017

Beckmann

Branch History Table

Fetch PC [| | 0 0]

I-Cache

k

BHT Index

2^k-entry BHT, 2 bits/entry

Instruction

Opcode   offset

+

Branch?   Target PC   Taken/¬Taken?

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder
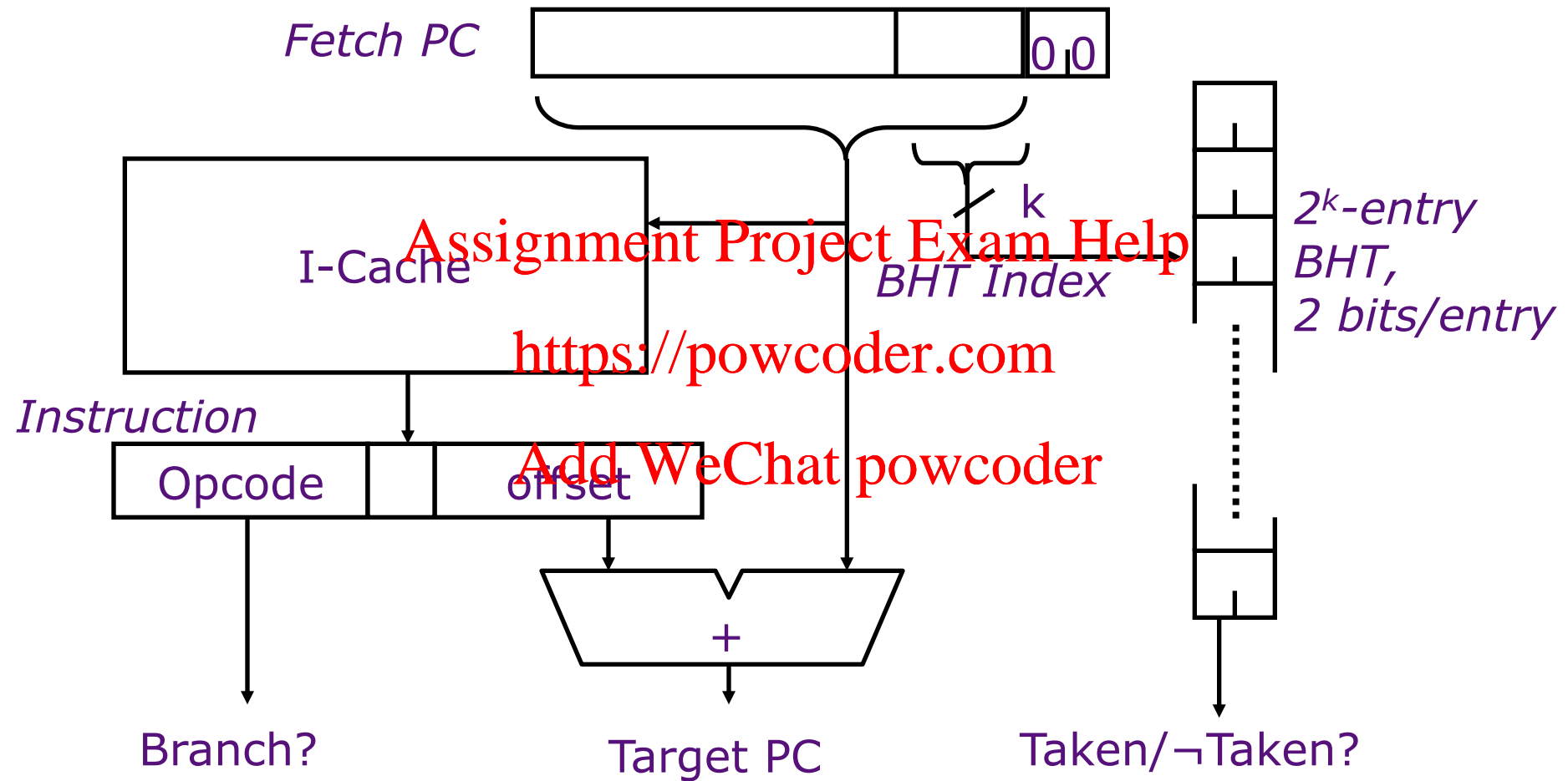
4K-entry BHT, 2 bits/entry, ~80-90% correct predictions

Beckmann

Exploiting Spatial Correlation
Yeh and Patt, 1992

```
if (x[i] < 7) then
        y += 1;
if (x[i] < 5) then
        c -= 4;
```
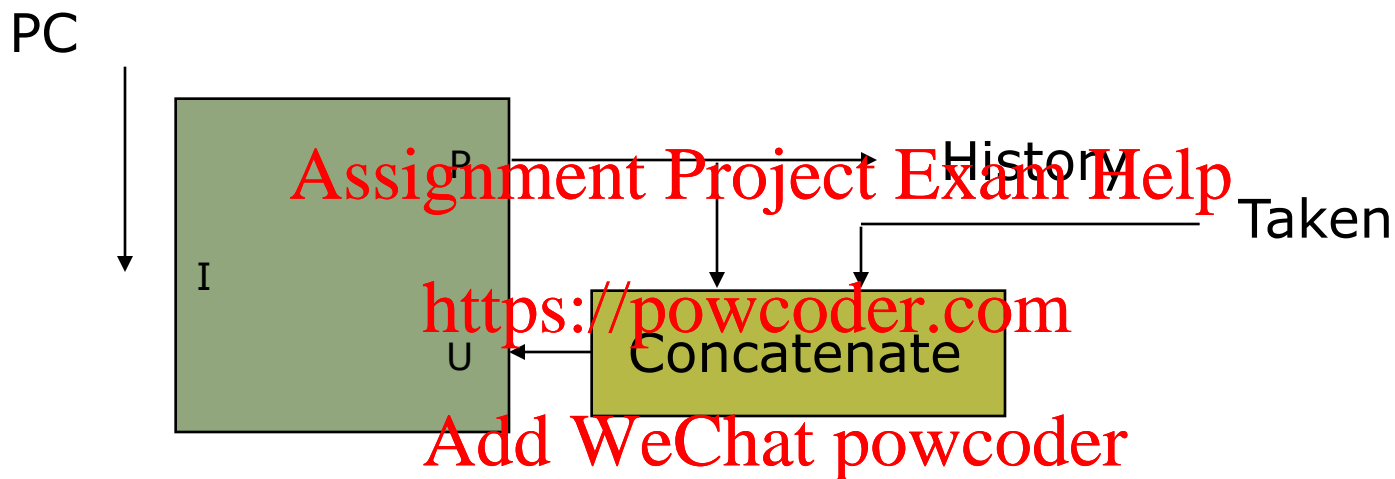
If first condition false, second condition also false

History register, H, records the direction of the last N branches executed by the processor

Beckmann

History Register

PC

P   History

I

U   Concatenate

Taken

History(PC, T) = P(PC; P || T)

Beckmann

Global History

Global History   Prediction

Concat

+/-

Taken

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder
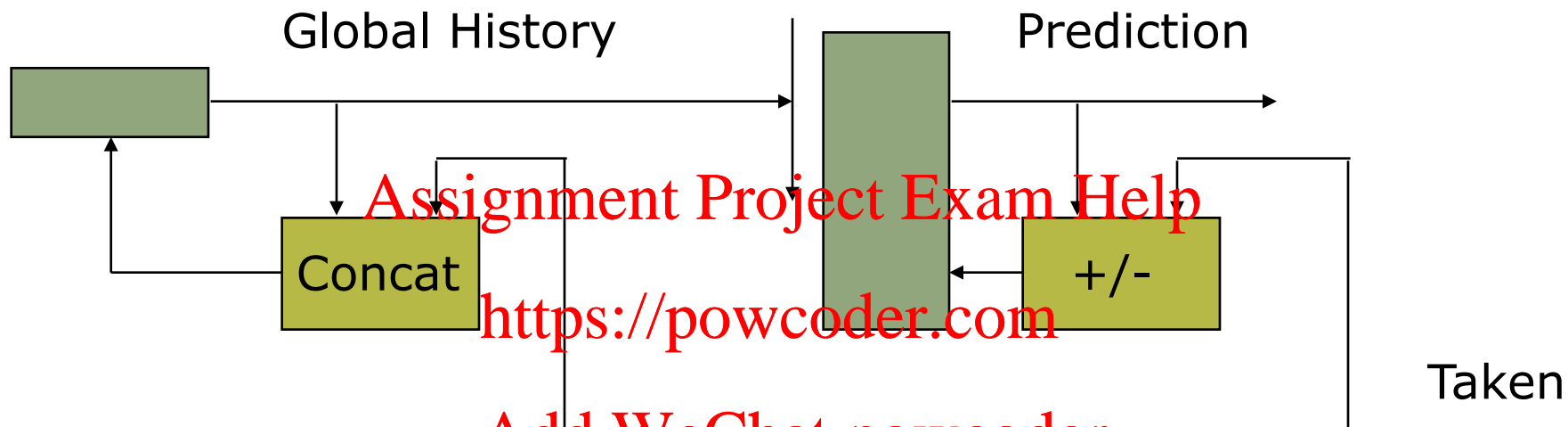
GHist(;T) = MSB(Counter(History(0, T); T))

Ind-Ghist(PC;T) = MSB(Counter(PC || Hist(GHist(;T);T)))

Can we take advantage of a pattern at a particular PC?

Beckmann

Local History

PC

Local History

Prediction

Concat

+/-

Taken
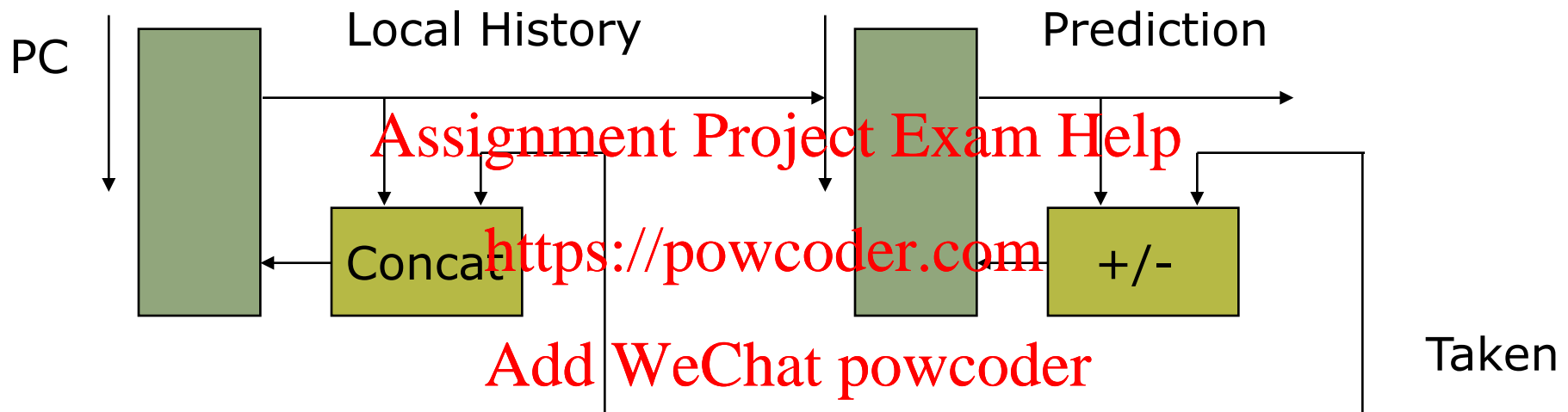
$$LHist(PC, T) = MSB(Counter(History(PC; T); T))$$

Can we take advantage of the global pattern at a particular PC?

Beckmann

Two-level Predictor

PC

Global
History

Prediction

Concat

Assignment Project Exam Help

Concat

https://powcoder.com

+/-

Add WeChat powcoder

Taken
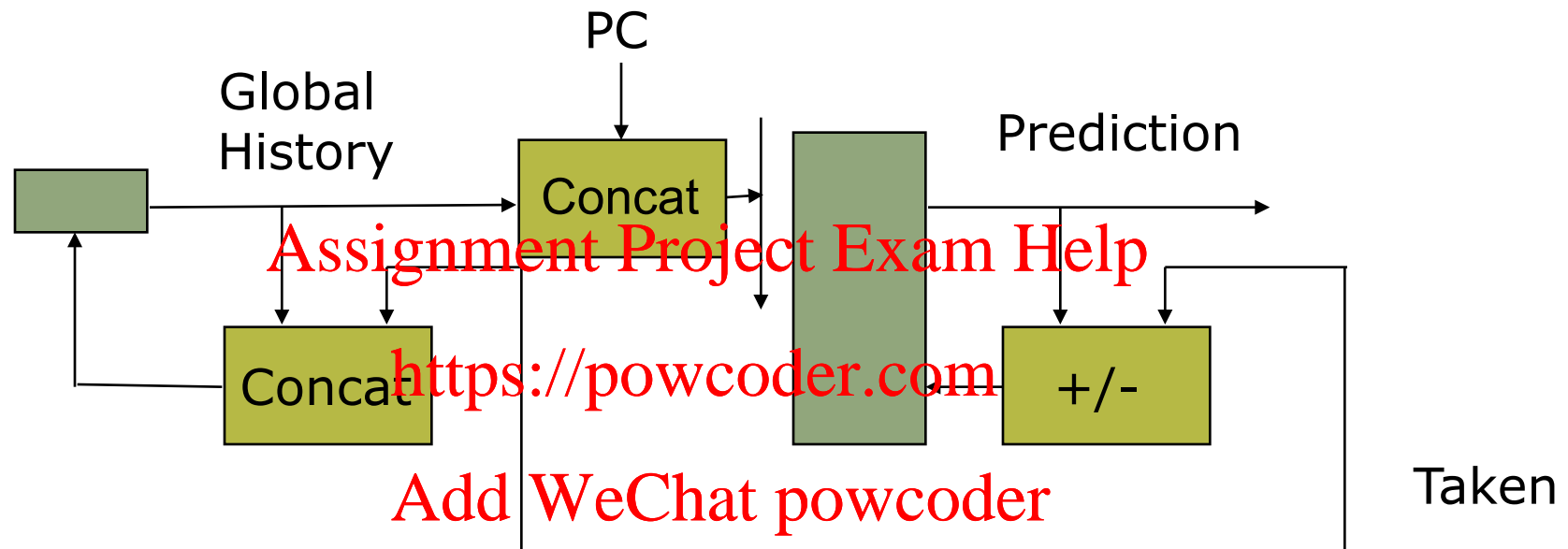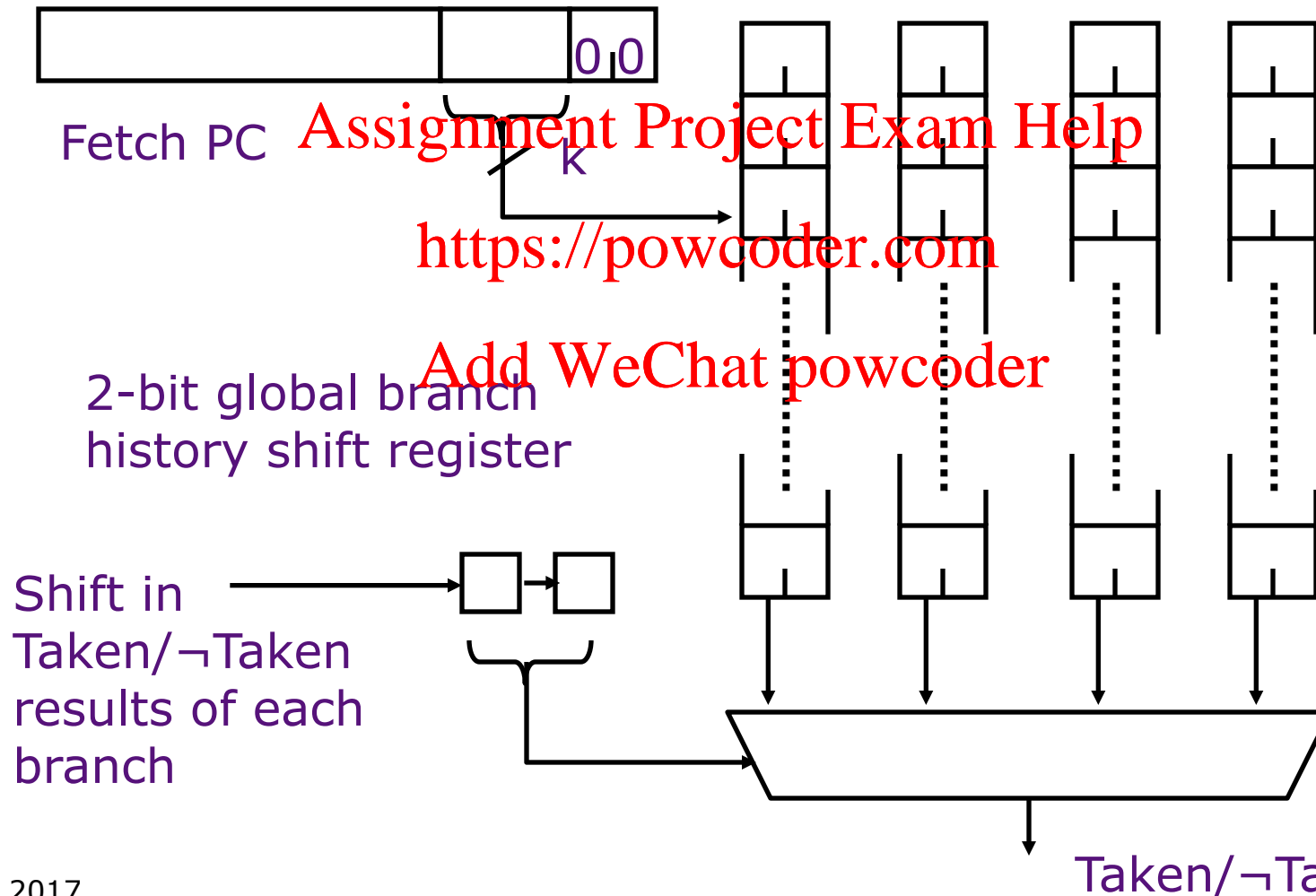
2Level(PC, T) = MSB(Counter(History(0; T)||PC; T))

Beckmann

Two-Level Branch Predictor

*Pentium Pro uses the result from the last two branches
to select one of the four sets of BHT bits (~95% correct)*

0 0

Fetch PC

K

2-bit global branch
history shift register

Shift in
Taken/¬Taken
results of each
branch

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

Taken/¬Taken?

Beckmann

Which predictor is best?

- Many different predictors were proposed

- Each handles particular patterns well

- Common principles: temporal / spatial correlation, saturating counters, etc

- But none is universal

- What to do?

Choosing Predictors

LHist

GHist

Chooser
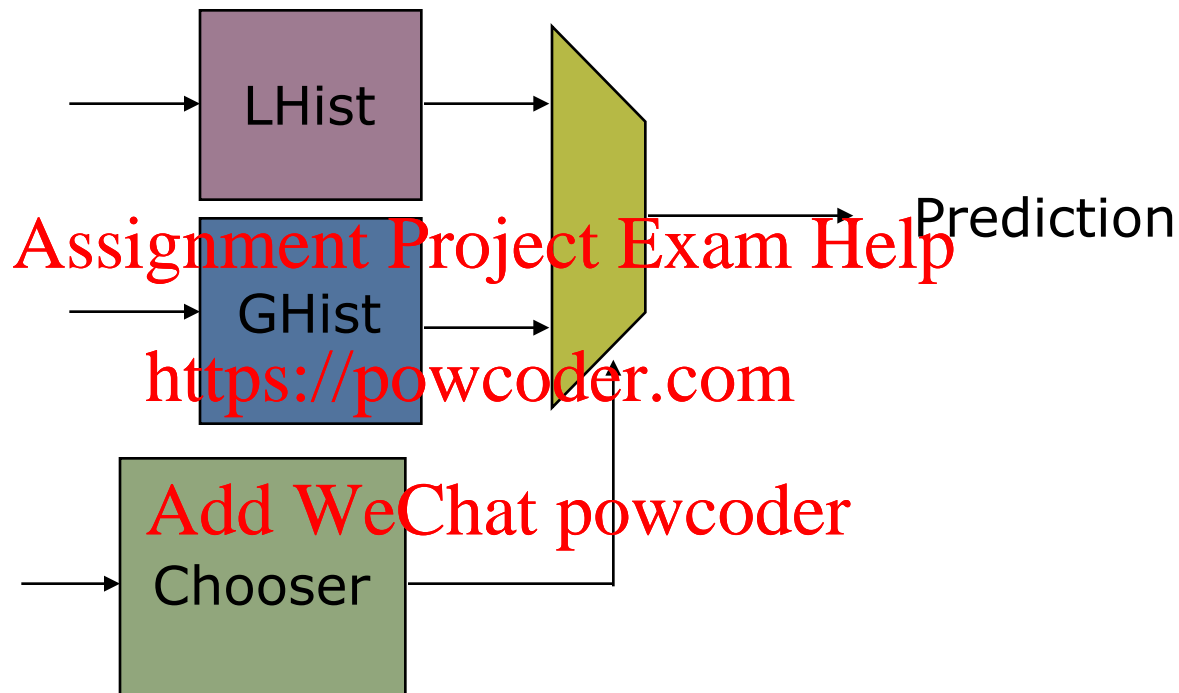
Prediction

Chooser = MSB(P(PC; P + (A==T) - (B==T)))
or
Chooser = MSB(P(GHist(PC; T); P + (A==T) - (B==T)))

Beckmann

Tournament Branch Predictor
(Alpha 21264)
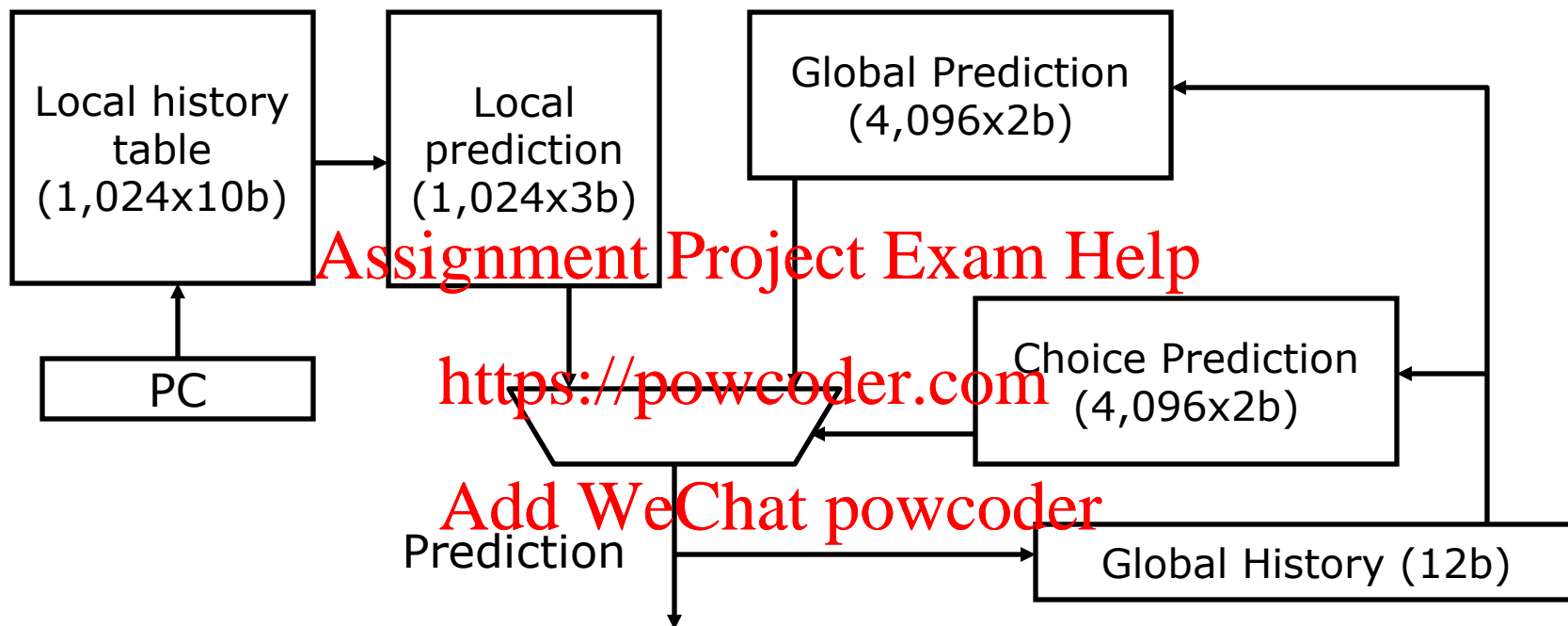
- Choice predictor learns whether best to use local or global branch history in predicting next branch
- Global history is speculatively updated but restored on mispredict
- Claim 90-100% success on range of applications

Beckmann

Sophisticated Designs

- Neural-network-based branch predictors
  - [Vintan, IJCNN '99][Jiminez, HPCA '01]
  - High prediction accuracy, but more computation
  - Actually implemented in AMD Piledriver, AMD Ryzen

- TAGE predictor
  - TAgged GEometric predictor [Seznac, JILP '06]
  - Keep multiple predictions with different history lengths
  - Partially tag predictions to avoid false matches
  - Only provide prediction on tag match
  - Rumored to be what Intel uses

Beckmann

TAGE predictor
Seznec & Michaud, 2006

| PC |

| BiModal | | TAGE[L1] | | TAGE[L2] | | TAGE[L3] |

Assignment Project Exam Help

https://powcoder.com

My Guess   Use me?

Add WeChat powcoder
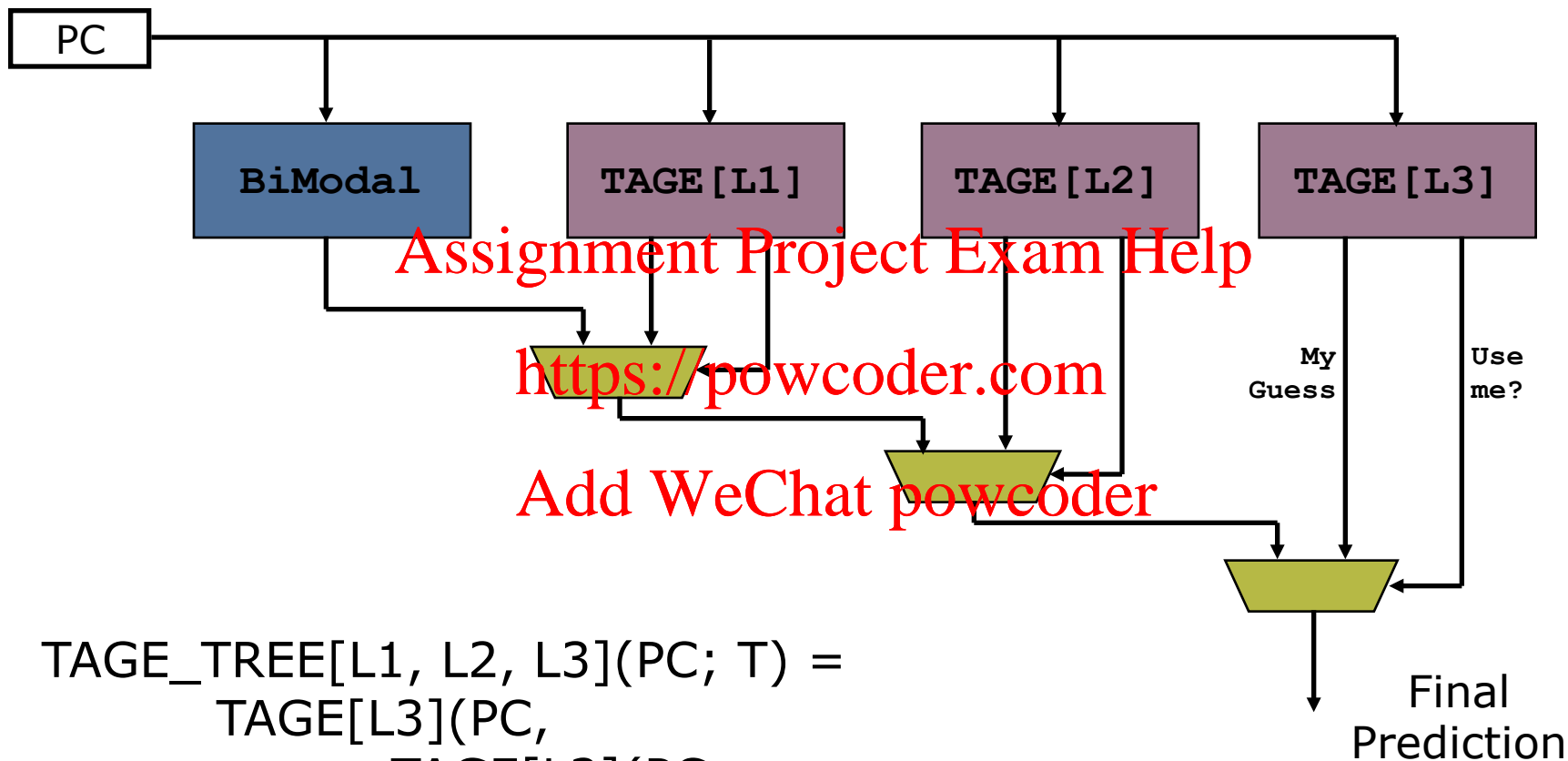
TAGE_TREE[L1, L2, L3](PC; T) =
        TAGE[L3](PC,
                TAGE[L2](PC,
                        TAGE[L1](PC, Bimodal(PC;T)
        ;T)   ;T   ;T)

Final Prediction

TAGE component

Next
Predictor

Counter

GHist

Useful

PC

Tag

Prediction

My
Guess

Use
me?

Useful bit is updated when predictor was correct and its alternative was wrong; used for replacement.

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

27 April 2017

Beckmann
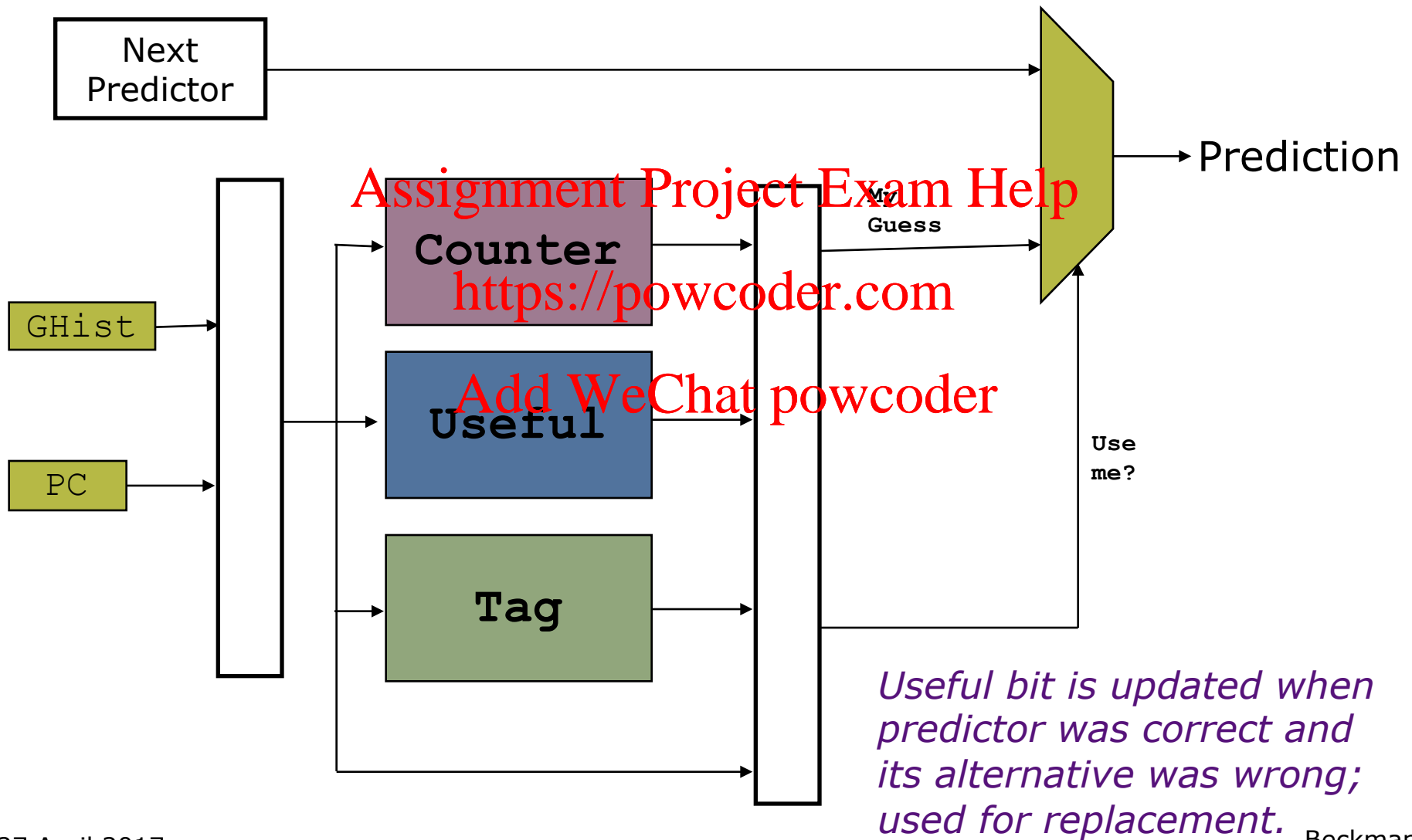
TAGE predictor component

TAGE[L](PC, NEXT; T) =

    idx = hash(PC, GHIST[L](;T))
    tag = hash(PC, GHIST[L](;T))

    TAGE.U = SA(idx, tag; ((TAGE == T) && (NEXT != T))?1:SA)
    TAGE.Counter = SA(idx, tag; T?SA+1:SA-1)

    use_me = TAGE.U && isStrong(TAGE.Counter)
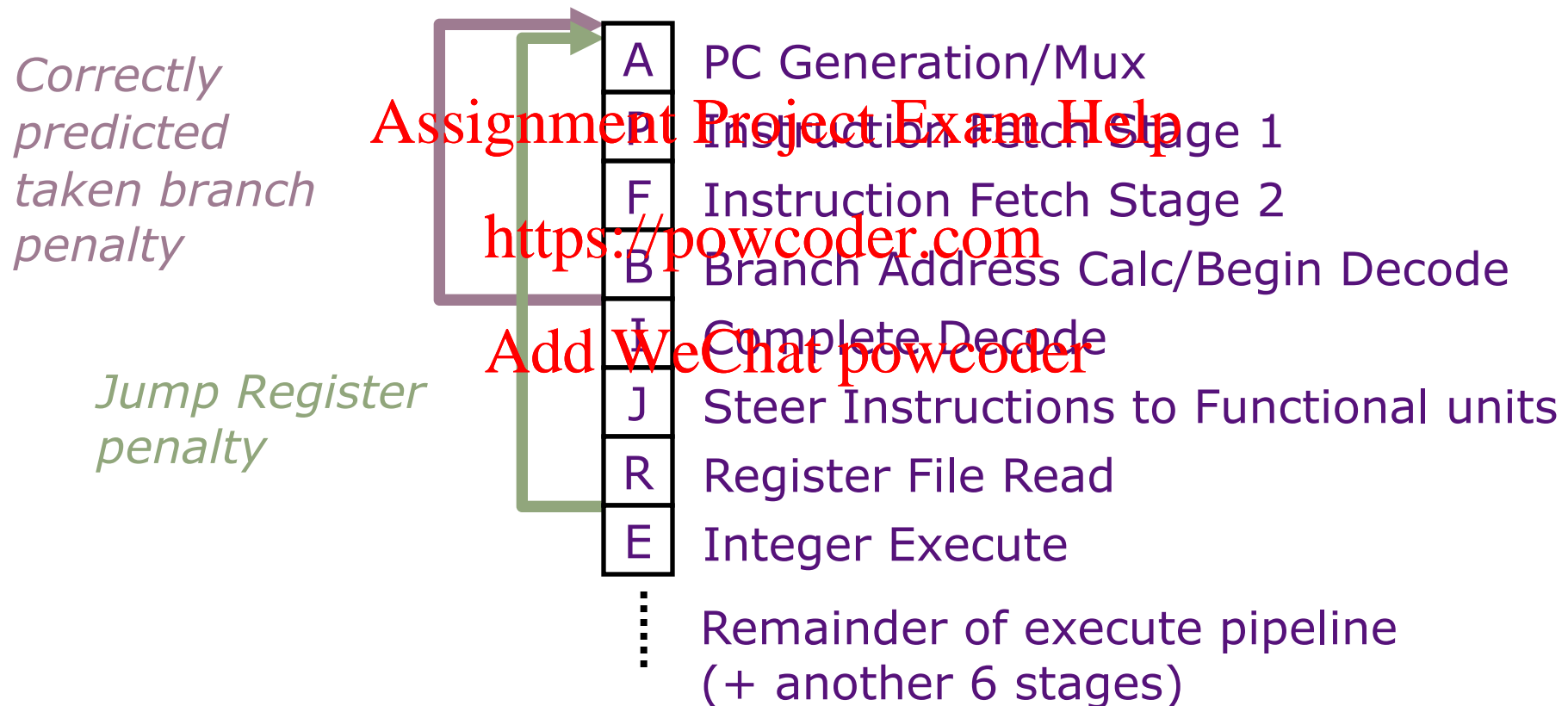   TAGE = use_me?MSB(TAGE.Counter):NEXT

    Notes:
        SA is a 'set associative' structure
        SA allocation occurs on mispredict (not shown)
        TAGE.U cleared on global counter saturation

Beckmann

Limitations of branch predictors

Only predicts branch direction. Therefore, cannot redirect fetch stream until after branch target is determined.

Correctly predicted taken branch penalty

Jump Register penalty

A	PC Generation/Mux
P	Instruction Fetch Stage 1
F	Instruction Fetch Stage 2
B	Branch Address Calc/Begin Decode
I	Complete Decode
J	Steer Instructions to Functional units
R	Register File Read
E	Integer Execute

Remainder of execute pipeline (+ another 6 stages)

UltraSPARC-III fetch pipeline

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

Beckmann

Branch Target Buffer (untagged)

IMEM

predicted
target

BPb

Branch
Target
Buffer
(BTB)
(2^k entries)

Assignment Project Exam Help

https://powcoder.com

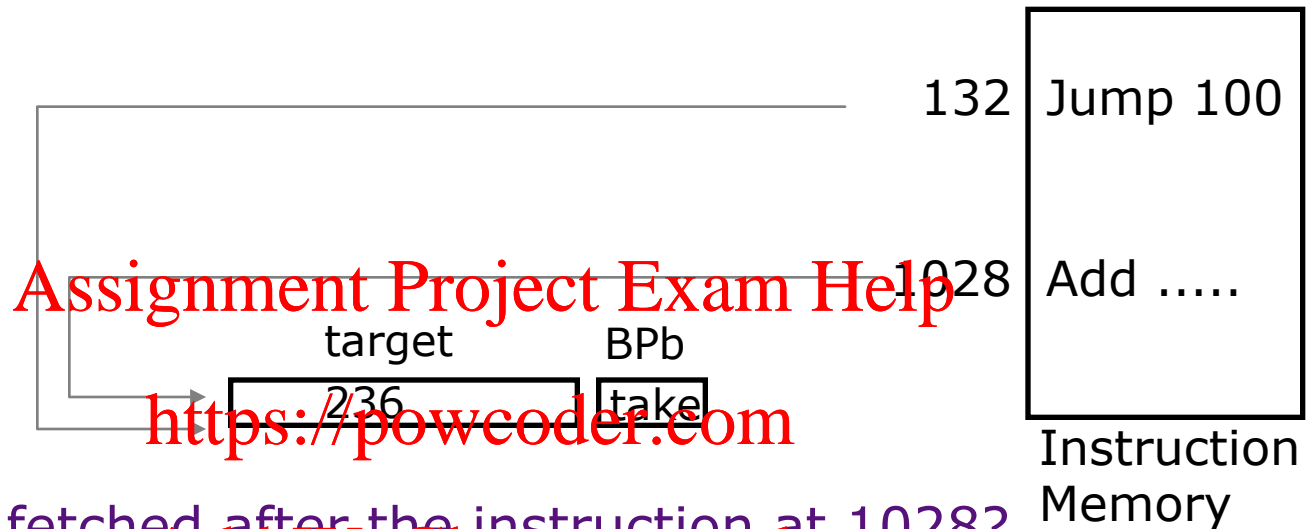k

PC

Add WeChat powcoder

target

BP
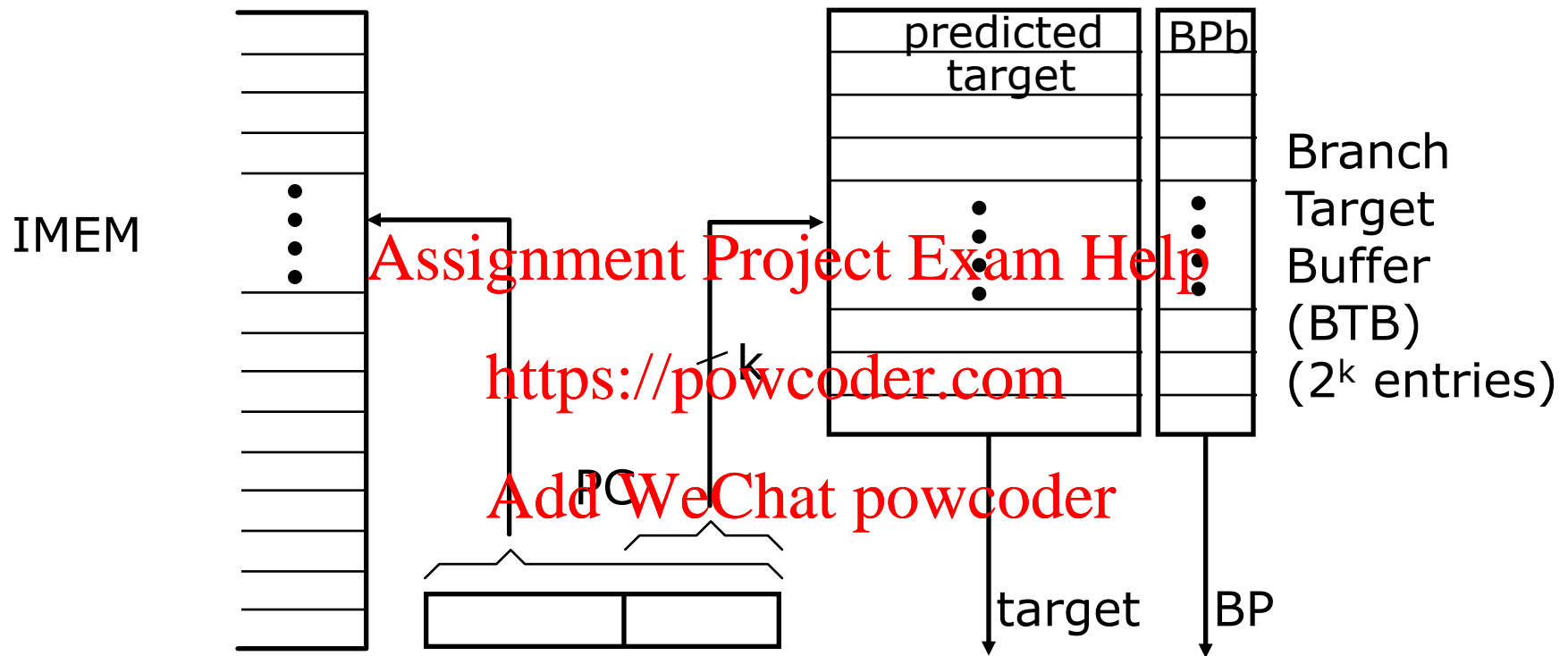
BP bits are stored with the predicted target address.

IF stage: *If (BP=taken) then nPC=target else nPC=PC+4*
later: *check prediction, if wrong then kill the instruction
            and update BTB & BPb else update BPb*

Beckmann

Address Collisions

*Assume a
128-entry
BTB*

132 | Jump 100

1028 | Add .....

Instruction
Memory

target   BPb

236 | take

What will be fetched after the instruction at 1028?

BTB prediction   =   236
Correct target   = 1032

$\Rightarrow$ *kill* PC=236 and *fetch* PC=1032

*Is this a common occurrence?
Can we avoid these bubbles?*

Beckmann

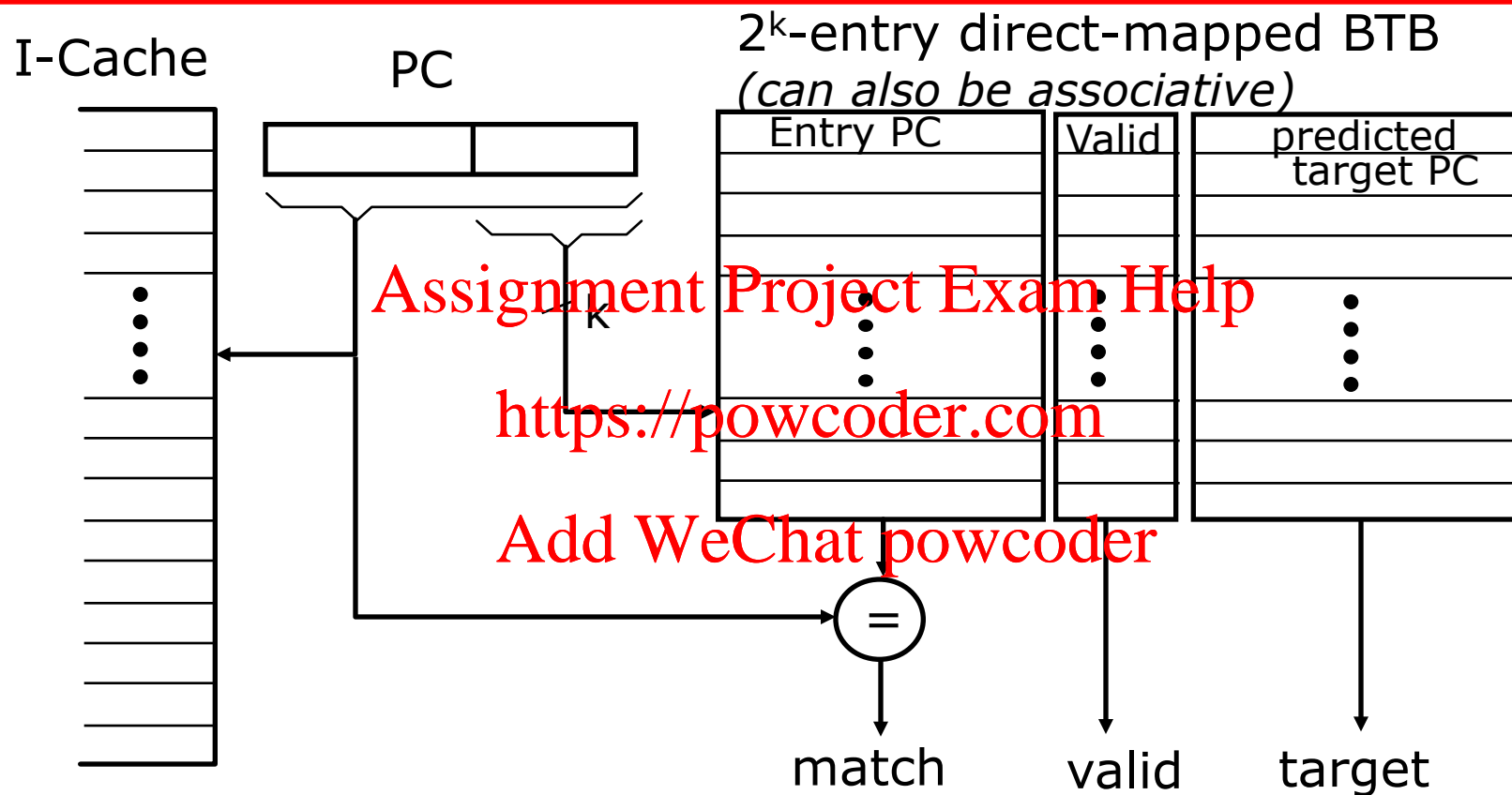# BTB is only for Control Instructions

BTB contains useful information for branch and jump instructions only

⇒ Do not update it for other instructions

For all other instructions the next PC is (PC)+4 !

How to achieve this effect without decoding the instruction?

Beckmann

Branch Target Buffer (tagged)

I-Cache

PC

2^k-entry direct-mapped BTB
(can also be associative)

Entry PC	Valid	predicted target PC

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

=

match   valid   target

- Keep both the branch PC and target PC in the BTB
- PC+4 is fetched if match fails
- Only *taken* branches and jumps held in BTB
- Next PC determined *before* branch fetched and decoded

Consulting BTB Before Decoding

	132	Jump 100

entry PC	target	BPb
132	236	take

	1028	Add .....
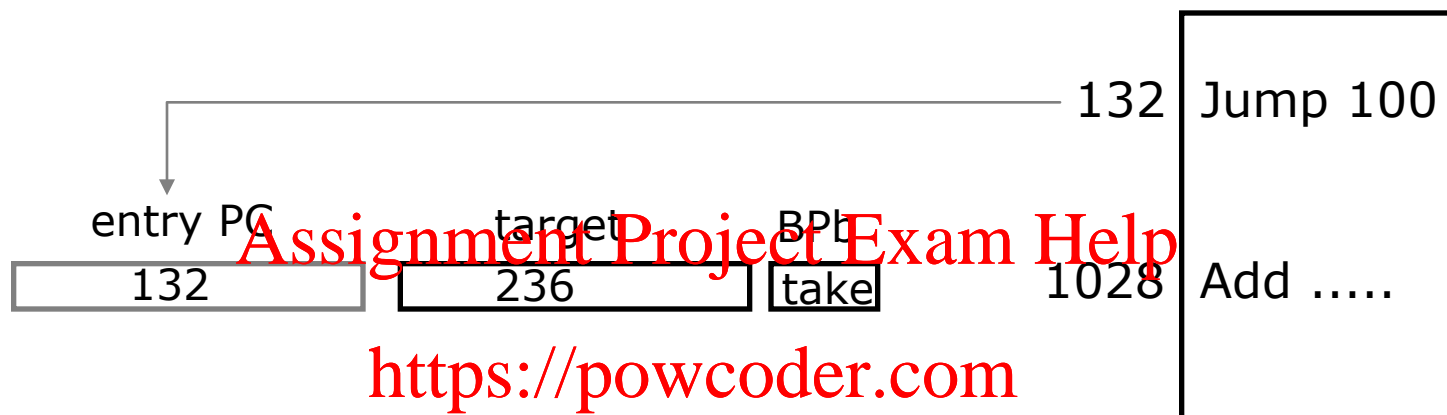
- The match for PC=1028 fails and 1028+4 is fetched
    ⇒ *eliminates false predictions after ALU instructions*

- BTB contains entries only for control transfer instructions
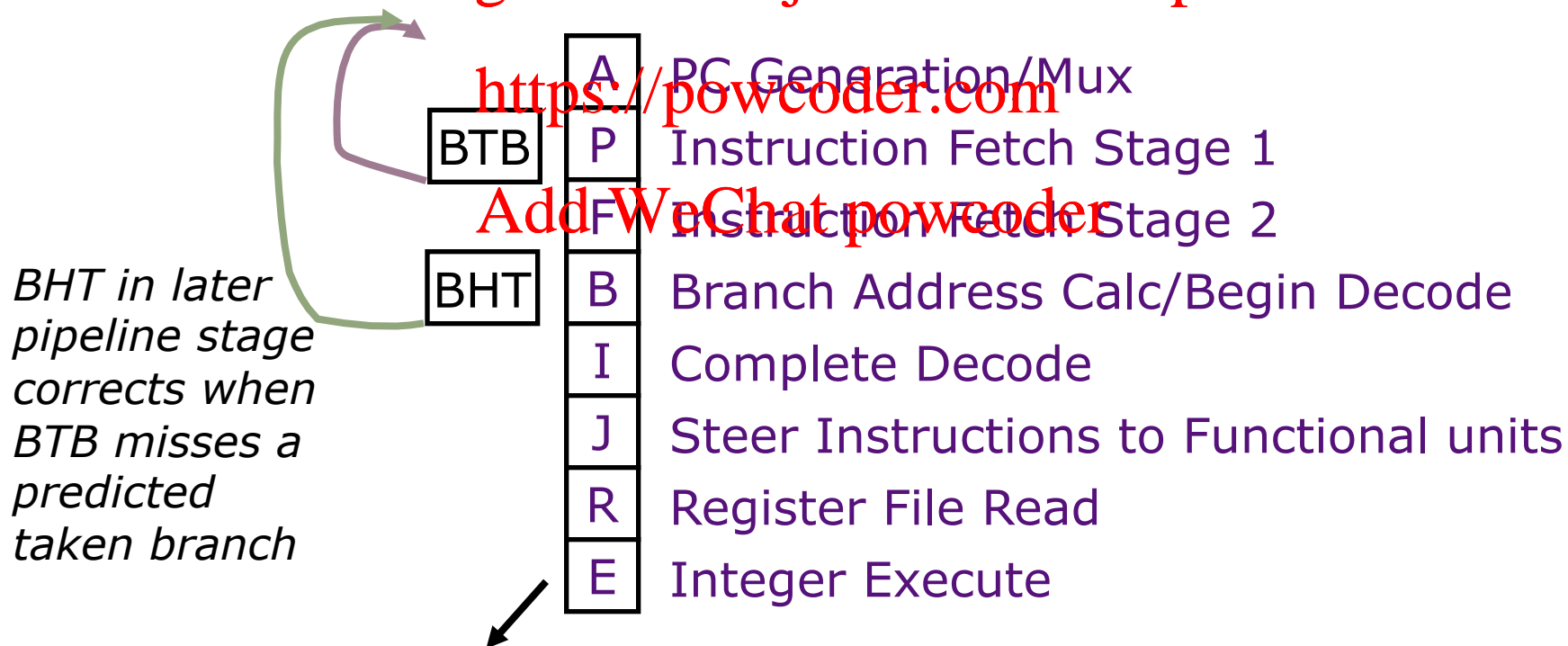   ⇒ *more room to store branch targets*

Beckmann

Combining BTB and BHT

- BTB entries are considerably more expensive than BHT, but can redirect fetches at earlier stage in pipeline and can accelerate indirect branches (JR)

- BHT can hold many more entries and is more accurate

	A	PC Generation/Mux
BTB	P	Instruction Fetch Stage 1
	F	Instruction Fetch Stage 2
BHT	B	Branch Address Calc/Begin Decode
	I	Complete Decode
	J	Steer Instructions to Functional units
	R	Register File Read
	E	Integer Execute

BHT in later pipeline stage corrects when BTB misses a predicted taken branch
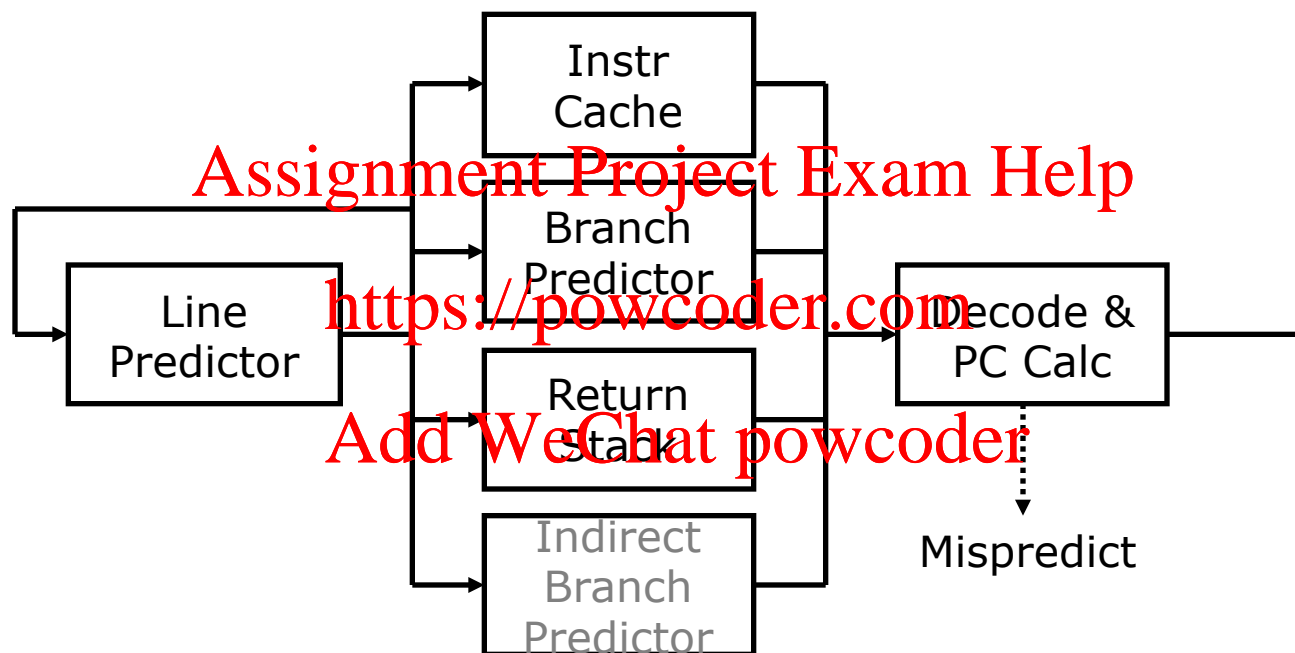
BTB/BHT only updated after branch resolves in E stage

Beckmann

Line Prediction
(*Alpha 21[234]64*)

- For superscalar useful to predict next cache line(s) to fetch

Instr Cache

Branch Predictor

Line Predictor

Return Stack

Decode & PC Calc

Indirect Branch Predictor

Mispredict

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

- Line Predictor predicts line to fetch each cycle (tight loop)
  - Untagged BTB structure – Why?
  - 21464 was to predict 2 lines per cycle
- Icache fetches block, and predictors improve target prediction
- PC Calc checks accuracy of line prediction(s)

Uses of Jump Register (JR)

- Switch statements (jump to address of matching case)

  BTB works well if same case used repeatedly

- Dynamic function call (jump to run-time function address)

  BTB works well if same function usually called, (e.g., in C++ programming, when objects have same type in virtual function call)

- Subroutine returns (jump to return address)

  BTB works well if usually return to the same place

  ⇒ *Often one function called from many distinct call sites!*

How well does BTB work for each of these cases?

Beckmann

Subroutine Return Stack

Small structure to accelerate JR for subroutine returns, typically much more accurate than BTBs.

```
fa() { fb(); }
```

```
fb() { fc(); }
```
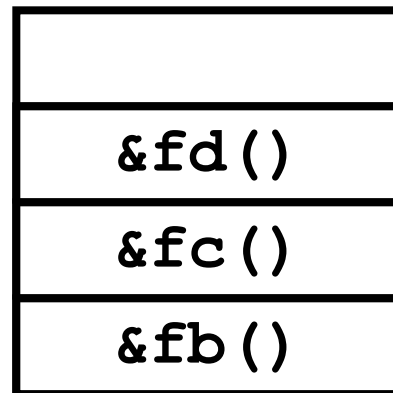
```
fc() { fd(); }
```

Push call address when function call executed

Pop return address when subroutine return decoded

&fd()
&fc()
&fb()

k entries (typically k=8-16)

Beckmann

Overview of branch prediction

Best predictors reflect program behavior

BTB

BP, JMP, Ret

Assignment Project Exam Help

https://powcoder.com

P C

Decode

Add WeChat powcoder

Reg Read

Execute

Need next PC immediately

Instr type, PC relative targets available

Simple conditions, register targets available

Complex conditions available

Tight loop

Loose loop

Loose loop

Loose loop
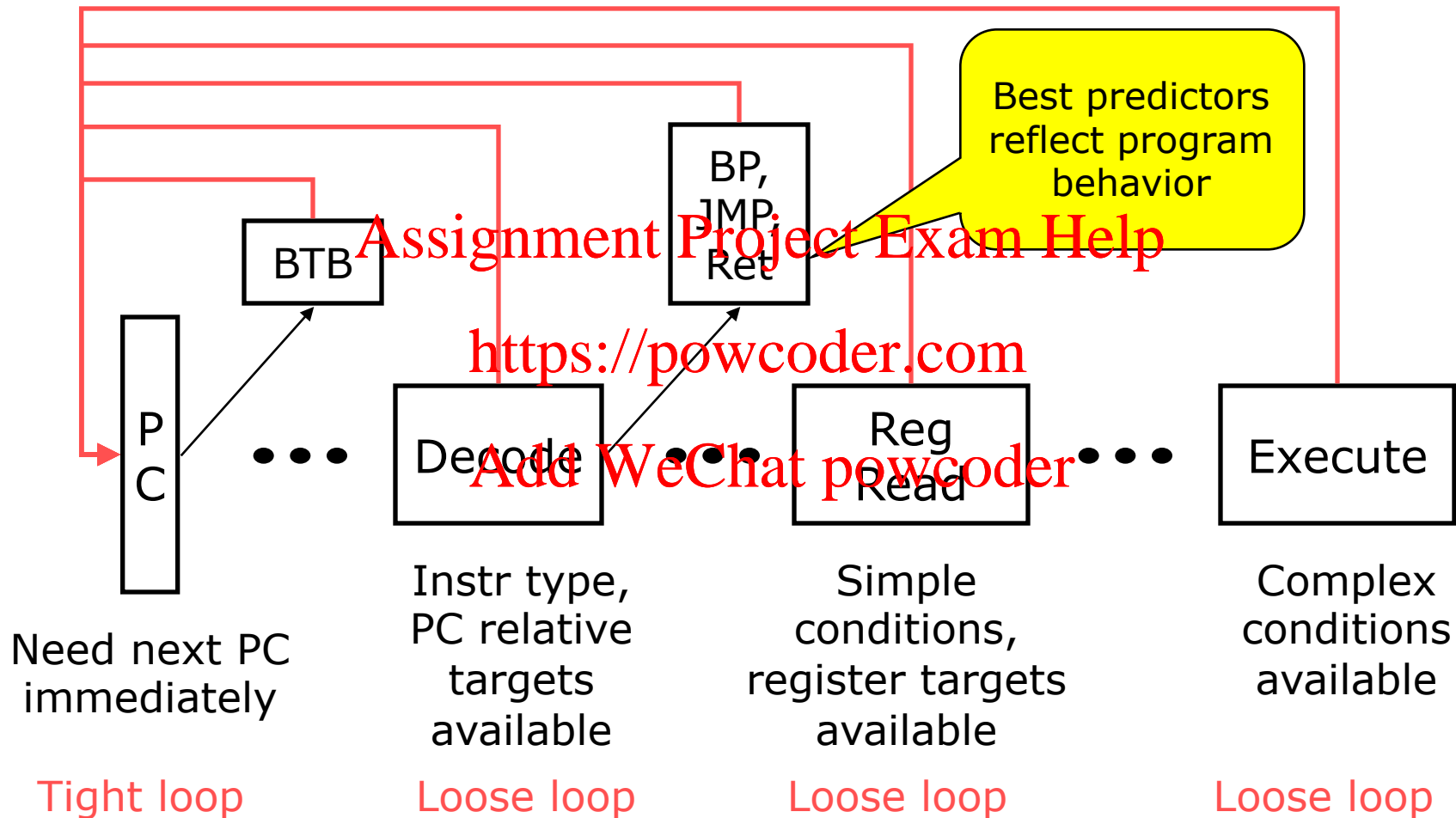
Must speculation check always be correct?   No...

Beckmann

Thank you !

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder