Theory Part

Thursday, February 17, 2022 10:35 AM

CDB:
- all Func Units write their result here as soon as an instruction is completely processed
- all reservation stations waiting for operand, register file, ROB read from here
- having multiple CDBs will allow multiple writes in one cycle, hence improving performance (but cost increased)

Loop Unrolling:
- static technique -> reduce # of iterations
- loop body modified by compiler

adv:
- # of branches reduced
- increased loop size -> increased chance of ILP identification

disadv:
- increased code size

Example

```
for (i=0;i<MAX;i++ ){
  y[i] = x[i]+ 5;
}
for (i=0;i<MAX;i+=4 ){
  y[i] = x[i]+ 5;
  y[i+1] = x[i+1]+ 5;
  y[i+2] = x[i+2]+ 5;
  y[i+3] = x[i+3]+ 5;
}
```

structural
resource conflicts
- pipeline unit can exec all op in given cc
- is 1 reg-file write port, req 2 reg writes
- is single-port mem, two instr access mem together

data
instr depending on res of prev instr
- dependence b/w instr and they are close
- fixed by fwding and stall

control
depend on branches and other instr that may change pc after the instr is fetched already
- stall at decode, decide early if branch taken or not, compute new PC value
- delayed branch

BTB:
1. Architecture
- n entries
- 2 x 32-bit fields i.e Address and Target
- 1 additional bit (optional)

2. Behaviour
- accessed on each Fetch
- entry selected -> log (n) bits [log is base 2]
- address field of entry & fetched instr compared; if matched, target field uploaded in PC
- btb possibly(if branch taken) updated when instr is completed

The BTB is initially empty (i.e., full of 0s)
The following instructions are executed in sequence

add.d f1,f2,f3	*located at the address 0x00A50050*
bnez r4,l1	*located at the address 0x00A50054; the branch is taken, and the branch target address is 0x00A60050*
mul.d f5,f6,f7	*located at the address 0x00A60050*
daddi r2,r2,-1	*located at the address 0x00A60054*
bez r2,l2	*located at the address 0x00A60058; the branch is not taken*
j l3	*located at the address 0x00A6005C the branch target address is 0x00A60AA0*

BTB final content:

Address	Target
0x00000000	0x00000000
0x00A50054	0x00A60050
0x00000000	0x00000000
0x00A6005C	0x00A60AA0

BHT
1. Architecture
- N elements of M bits each (1 or 2)

2. Behaviour
- Accessed at decode stage of a condtional branch instr
- using log (n) bits while ignoring alignment bits
- entry predicts taken or notTaken
- based on previous execution of the same instr
- if res of branch is known, BHT is possibly updated

4 entries so 2 bits will be used to select the entry (except last 2)

Assuming that the processor uses 32 bit addresses, each instruction is 4 byte wide, and the BHT is composed of 16 entries, identify the final content of the BHT if
- The BHT is initially empty (i.e., full of 0s, meaning NotTaken)
- The following instructions are executed in sequence
  - l.d r2,v1(r1) located at the address 0x00A50050
  - bnez r2,l1 located at the address 0x00A50054; the branch is taken, and the branch target address is 0x00A60050
  - addi r2, r2, #1 located at the address 0x00A60050
  - bez r2,l2 located at the address 0x00A60054; the branch is not taken
  - andi r5, r5, #1 located at the address 0x00A60058
  - bez r2,l3 located at the address 0x00A6005C; the branch is taken, and the branch target address is 0x00B80040
  - add r1, r2, r3 located at the address 0x00B80040
  - bez r1,l4 located at the address 0x00B80044; the branch is taken, and the branch target address is 0x00A50050.

- 54 -> 0101 0100 -> entry 5 -> already 0 (misprediction) -> actually taken, so new value 1
- 54 -> 0101 0100 -> entry 5 -> already 1 (misprediction) -> actually not Taken, so new value 0
- 5C -> 0101 1100 -> entry 7 -> already 0 (misprediction) -> actually taken, so new value 1
- 44 -> 0100 0100 -> entry 1 -> already 0 (misprediction) -> actually taken, so new value 1

0
1
0
0
0
0
1
-
-

ROB
The ROB is crucial to allow in-order instruction commitment while execution is performed out-of-order
Composition
elements of the buffer are composed of the following fields:
- o Instruction type
- o Destination
- o Value
- o Ready

Working
- An entry in the ROB is allocated to an instruction when this is successfully issued.
- Its content is written via the CDB when the execution finishes its execution.
- It is read when other instructions use the result produced by the instruction while this is not yet committed.
- The ROB is a FIFO (i.e., a circular buffer). When an instruction becomes the oldest in the ROB, it is committed (i.e., its result is written in the destination) and the entry is de-allocated.
- If the instruction to be committed is a mispredicted branch, the whole buffer is flushed.

Advantage
The ROB allows the processor to implement dynamic scheduling with speculation. It also allows precise exception management.

cc:
add 2
mul 10
div 40
ld 2

Instruction status

Instruction	j	k	Issue	Execution complete	Write Result
LD F6	34+	R2	1	3	4
LD F2	45+	R3	2	5	6
MULT F0	F2	F4	3	16	17
SUBD F8	F6	F2	4	8	9
DIVD F10	F0	F6	5	58	59
ADDD F6	F8	F2	6	12	13

```
1   L.D     F6, 34(R2)
2   L.D     F2, 45(R3)
3   MUL.D   F0, F2, F4
4   SUB.D   F8, F2, F6
5   DIV.D   F10, F0, F6
6   ADD.D   F6, F8, F2
```

Assumptions:
- Add takes 2 clock cycles, Multiply 10, Divide 40
- Loads requires 2 ccs

-!
Ld str exe !o >issue
Mem X, ALU, Branch, Store
CDB X, Branch Store
Exe+1 in mem
ALU ki CDB

Exe #1

if mul 6
if div 8
if store 2

# iteration		ISSUE	EXE	MEM	CDB	COMMIT
1	l.d f1,v1(r1)	1	2	3	4	5
1	l.d f2,v2(r1)	1	3	4	5	
1	l.d f3,v3(r1)	2	4	5	6	
1	div.d f5, f3, f11	2	7	X	15	
1	sub.d f4, f1, f2	3	6	X	8	
1	add.d f6, f4, f5	3	16	X	18	
1	div.d f7,f6,f12	4	23	X	31	
1	s.d f7,v4(r1)	4	5	X	X	
1	daddui r1,r1,8	5	6	X	9	
1	daddi r2,r2,-1	5	7	X	9	
1	bnez r2,loop	6	10	X	X	
2	l.d f1,v1(r1)	7	8	9	10	
2	l.d f2,v2(r1)	7	9	10	11	
2	l.d f3,v3(r1)	8	10	11	12	
2	div.d f5, f3, f11	8	15	X	23	
2	sub.d f4, f1, f2	9	12	X	14	
2	add.d f6, f4, f5	9	24	X	26	
2	div.d f7,f6,f12	10	31	X	39	
2	s.d f7,v4(r1)	10	11	X	X	
2	daddui r1,r1,8	11	12	X	13	
2	daddi r2,r2,-1	11	13	X	16	
2	bnez r2,loop	12	17	X	X	

pipelined div continues from the same cycle as in CDB, all others from CDB+1

->issue q first row
-> exe of ld & sd
  -> sequence in column
  -> greater than issue
->Mem 'X' : ALU, Store, Branch
->CDB 'X' : Store, Branch
-> Exe +1 in MEM & MEM+1 in CDB
-> F.P ALU : EXE & CDB
-> Other ALU & Branch.

For Load Instructions

# iteration		ISSUE	EXE	MEM	CDB	COMMIT	Notes
1	l.d f1,v1(r1)	1	2m	3	4	5	
1	l.d f2,v2(r1)	1	3m	4	5	6	
1	l.d f3,v3(r1)	2	4m	5	6	7	
1	div.d f5, f3, f11	2	7d		15	16	Wait for f3
1	sub.d f4, f1, f2	3	6a		8	17	Wait for f2
1	add.d f6, f4, f5	3	16a		18	19	Wait for f5
1	div.d f7,f6,f12	4	23d		31	32	Wait for f6, then for div unit
1	s.d f7,v4(r1)	4	5m			33	
1	daddui r1,r1,8	5	6i		7	34	
1	daddi r2,r2,-1	5	7i		9	35	
1	bnez r2,loop	6	10j			36	Wait for r2
2	l.d f1,v1(r1)	7	8m	9	10	37	
2	l.d f2,v2(r1)	7	9m	10	11	38	
2	l.d f3,v3(r1)	8	10m	11	12	39	
2	div.d f5, f3, f11	8	15d		23	40	Wait for f3, then for div unit
2	sub.d f4, f1, f2	9	12a		14	41	Wait for f2
2	add.d f6, f4, f5	9	24a		26	42	Wait for f5
2	div.d f7,f6,f12	10	31d		39	43	Wait for f6, then for div unit
2	s.d f7,v4(r1)	10	11m			44	
2	daddui r1,r1,8	11	12i		13	45	
2	daddi r2,r2,-1	11	13i		16	46	
2	bnez r2,loop	12	17j			47	Wait for r2

Feature	ARM LPC1768	Intel 8086
Architecture	RISC	CISC
Data bus width	32 bits	16 bits
Endianness	Little-endian	Little-endian
Clock speed	Up to 100 MHz	Up to 10 MHz
Memory addressing	Up to 64 MB	Up to 1 MB
Register file size	16 general-purpose registers	8 general-purpose registers
Interrupt handling	Uses vectored interrupts	Uses interrupt vector table
I/O interface	Uses GPIO pins for I/O	Uses separate I/O address space

Feature	MIPS64	MIPS32	MIPS16
Architecture	64-bit	32-bit	16-bit
Word size	64 bits	32 bits	16 bits
Registers	32 general-purpose, 32 floating-point	32 general-purpose, 32 floating-point	16 general-purpose
Address space	2^64 bytes	2^32 bytes	2^16 bytes
Instruction set	Large and complex	Reduced and simplified	Further reduced and simplified
Performance	High performance	Good performance	Lower performance
Application	High-end computing	Embedded systems, routers, networking equipment	Low-end embedded systems, microcontrollers

Feature	CISC	RISC
Philosophy	Complex instruction set	Reduced instruction set
Instruction length	Variable	Fixed
Number of instructions	Large	Small
Execution cycle	One cycle per instruction	Multiple cycles per instruction
Execution time	Longer execution time per instruction	Shorter execution time per instruction
Register usage	Memory-to-memory operations	Register-to-register operations
Memory access	Direct memory access	Load/store architecture
Pipeline architecture	More difficult to pipeline	Easier to pipeline
Code density	High	Low
Power consumption	Higher	Lower
Instruction set	Rich and complex	Simplified and limited
Applications	Desktops, servers, mainframes	Embedded systems, mobile devices

Here are some key points about the address bus:

1. **Memory Addressing**: The primary function of the address bus is to facilitate memory addressing. It allows the CPU (Central Processing Unit) to specify the exact location in the computer's memory where data or instructions are to be read from or written to.
2. **Size**: The width of the address bus determines the maximum number of unique memory locations or addresses that the computer can access. For example, a 16-bit address bus can address 2^16 (65,536) different memory locations, while a 32-bit address bus can address 2^32 (approximately 4.3 billion) different locations.
3. **Binary Representation**: Addresses on the address bus are represented in binary format. Each wire or line in the address bus carries a binary digit (bit), and the combination of bits on all the lines forms the binary address.
4. **Addressable Memory**: The size of the address bus also dictates the maximum amount of addressable memory. For example, a 32-bit address bus can address up to 4 GB (gigabytes) of memory because it can specify 2^32 different memory locations.
5. **Expansion**: In systems with larger memory requirements, the width of the address bus may be expanded to accommodate more memory. For instance, 64-bit processors have a 64-bit address bus, allowing them to address much larger memory spaces.
6. **Address Decoding**: The computer's memory management hardware uses the address bus to decode memory addresses and route data requests to the appropriate memory cells or storage locations.
7. **Input/Output (I/O) Addressing**: In addition to memory addressing, the address bus can also be used for addressing I/O devices and ports. This allows the CPU to communicate with peripherals and external hardware components.
8. **Control Signals**: The address bus is often accompanied by control signals, which include read and write signals, to indicate the direction of data transfer and control memory access.
9. **Compatibility**: The width of the address bus is an important factor in determining the compatibility of hardware components. For example, a CPU with a 32-bit address bus is typically compatible with memory modules designed for 32-bit addressing.

Aspect	Monolithic Architecture	Microkernel Architecture
Architecture Philosophy	Single tightly integrated unit	Modular design with core services in kernel
Complexity	More complex	Less complex
Reliability and Stability	Prone to system-wide failures	More resilient to failures
Performance	Lower communication overhead	Potentially higher communication overhead
Flexibility and Extensibility	Challenging to extend and modify	Easier to add/replace components

Aspect	Superscalar Architecture	Multi-core Architecture
Parallel Execution	Executes multiple instructions within a single core in parallel during a clock cycle.	Has multiple independent processor cores that execute instructions simultaneously.
Instruction Dispatch	Dispatches multiple instructions to different execution units within a single core.	Dispatches different threads or tasks to separate cores, each with its own resources.
Independence	Parallel execution limited to a single core; instructions must be independent within that core.	True parallelism achieved by having multiple independent cores, allowing for concurrent execution.
Usage Scenario	Effective for improving single-threaded performance and instruction throughput within a single core.	Effective for handling multiple threads or tasks simultaneously, benefiting multi-threaded applications and multitasking.

1. **Associative Memory**:
   - Associative memory, also known as content-addressable memory (CAM), works by searching for data based on its content rather

Data Bus:

0 2 2 1

Address	Code	BHT	Iteration #1		Iteration #2		Iteration #3		Iteration #4	
		entry #	prediction	result	prediction	result	prediction	result	prediction	result
0x0000	daddui r1, r0, 1									
0x0004	daddui r2, r0, 0									
0x0008	daddui r3, r0, 0									
0x000c	daddui r4, r0, 0									
0x0010	daddui r5, r0, 6									
0x0014	cyc: beq r2, r5, term									
0x0018	lb r6, vec(r2)									
0x001c	beq r6, r0, eq0									
0x0020	beq r6, r1, eq1									
0x0024	j nxt									
0x0028	eq0: daddui r3, r3, 1									
0x002c	j nxt									
0x0030	eq1: daddui r4, r4, 1									
0x0034	nxt: daddui r2, r2, 1									

c 63 o 75 d 64 i 69

Address	Code	BHT	Iteration #1		Iteration #2		Iteration #3		Iteration #4	
		entry #	prediction	result	prediction	result	prediction	result	prediction	result
0x0000	daddui r1, r0, 2									
0x0004	daddi r2, r0, -1									
0x0008	daddui r5, r0, 0									
0x000c	daddui r6, r0, 0									
0x0010	cyc: daddi r2, r2, 1									
0x0014	lb r3, vec(r2)									
0x0018	beqz r3, term	6	NT	NT	NT	NT	NT	NT	NT	NT
0x001c	daddu r4, r3, r0									
0x0020	ddivu r3, r3, r1									
0x0024	dmultu r3, r3, r1									
0x0028	dsubu r4, r4, r3									
0x002c	bnez r4, nxt	3	NT	T	T	T	T	NT	NT	T
0x0030	daddui r5, r5, 1									
0x0034	j cyc									
0x0038	nxt: beqz r4, cyc	6	NT	NT	NT	NT			NT	NT
0x003c	daddui r6, r6, 1									
0x0040	j cyc									
0x0044	term: sb r5, rese(r0)									
0x0048	sb r6, reso(r0)									
0x004c	halt									

n 74 g 67

compatibility of hardware components. For example, a CPU with a 32-bit address bus is typically compatible with memory modules designed for 32-bit addressing.

1. **Associative Memory**:
   - Associative memory, also known as content-addressable memory (CAM), works by searching for data based on its content rather than its address.
   - It stores data along with associated tags or keywords.
   - When you query the memory with a specific content or keyword, it searches through the stored data to find a match, returning the associated data.
   - This type of memory is often used in applications like caching and hardware routing tables to quickly locate information based on its content.
2. **Cross Compiler**:
   - A cross compiler is a software tool that generates executable code for a platform different from the one on which the compiler itself runs.
   - It is commonly used in embedded systems and cross-development environments.
   - Cross compilers are necessary when you want to develop software for a target platform (e.g., an embedded system or a different architecture) that is not compatible with the host system where you write the code.
   - The cross compiler translates the source code into machine code that can run on the target platform, allowing developers to create software for various environments without needing a dedicated compiler for each.

Usage Scenario	Effective for improving single-threaded performance and instruction throughput within a single core.	Effective for handling multiple threads or tasks simultaneously, benefiting multi-threaded applications and multitasking.

Data Bus:

- The data bus is another critical component of a computer's system bus alongside the address bus and control bus.
- It is a set of parallel lines or wires that allow data to be transferred between various parts of the computer, such as the CPU, memory, and peripherals.
- The width of the data bus determines how many bits of data can be transferred simultaneously. For example, a 32-bit data bus can carry 32 bits of data at a time.
- The data bus is bidirectional, meaning it can carry data in both read and write directions, facilitating the flow of information within the computer.
- Data buses play a central role in tasks like reading data from memory, writing data to memory, exchanging information with peripherals, and performing arithmetic and logic operations.

Control Bus:

- The control bus, like the data bus, is a set of parallel lines or wires within the computer's system bus.
- Its purpose is to carry control signals and commands that govern the operation of various components within the computer system.
- Control signals on the control bus include signals like "read," "write," "interrupt request," "clock," and various control codes that coordinate activities between the CPU, memory, and peripherals.
- The control bus plays a crucial role in activities such as fetching instructions from memory, signaling when data is ready to be read from or written to memory, and managing input/output operations.
- Control signals on the control bus help ensure that the computer's components work together seamlessly, allowing for orderly and synchronized data flow and operation.

```
0x001c      beq r6, r0, eq0
0x0020      beq r6, r1, eq1
0x0024      j nxt
0x0028  eq0:  daddui r3, r3, 1
0x002c      j nxt
0x0030  eq1:  daddui r4, r4, 1
0x0034  nxt:  daddui r2, r2, 1
0x0038      j cyc
0x003c  term:  sb r3, res0(r2)
0x0040      sb r4, res1(r2)
0x0044  halt
```

Address	Code	BHT	Iteration #5		Iteration #6		Iteration #7		Iteration #8	
		entry #	prediction	result	prediction	result	prediction	result	prediction	result
0x0000	daddui r1, r0, 1									
0x0004	daddui r2, r0, 0									
0x0008	daddui r3, r0, 0									
0x000c	daddui r4, r0, 0									
0x0010	daddui r5, r0, 6									
0x0014	cyc: beq r2, r5, term									
0x0018	lb r6, vec(r2)									
0x001c	beq r6, r0, eq0									
0x0020	beq r6, r1, eq1									
0x0024	j nxt									
0x0028	eq0: daddui r3, r3, 1									
0x002c	j nxt									
0x0030	eq1: daddui r4, r4, 1									
0x0034	nxt: daddui r2, r2, 1									
0x0038	j cyc									
0x003c	term: sb r3, res0(r2)									
0x0040	sb r4, res1(r2)									
0x0044	halt									

mis-predicted	

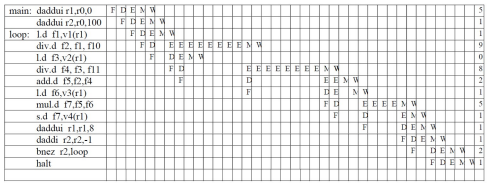
final content	
0	
1	
2	
3	
4	
5	
6	
7	

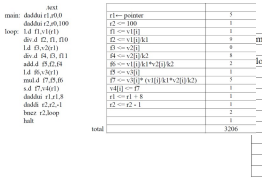
```
0x002c      bnez r4, nxt
0x0030      daddui r5, r5, 1
0x0034      j cyc
0x0038  nxt:  beqz r4, cyc
0x003c      daddui r6, r6, 1
0x0040      j cyc
0x0044  term:  sb r5, reset(r0)
0x0048      sb r6, reset(r0)
0x004c  halt
```

Address	Code	BHT	Iteration #5		Iteration #6		Iteration #7		Iteration #8	
		entry #	prediction	result	prediction	result	prediction	result	prediction	result
0x0000	daddui r1, r0, 2									
0x0004	daddi r2, r0, -1									
0x0008	daddui r5, r0, 0									
0x000c	daddui r6, r0, 0									
0x0010	cyc: daddui r2, r2, 1									
0x0014	lb r3, vec(r2)									
0x0018	beqz r3, term									
0x001c	daddi r4, r3, r0									
0x0020	ddivu r3, r3, r1									
0x0024	dmuln r3, r3, r1									
0x0028	daddu r4, r4, r3									
0x002c	bnez r4, nxt									
0x0030	daddui r5, r5, 1									
0x0034	j cyc									
0x0038	nxt: beqz r4, cyc									
0x003c	daddui r6, r6, 1									
0x0040	j cyc									
0x0044	term: sb r5, reset(r0)									
0x0048	sb r6, reset(r0)									
0x004c	halt									

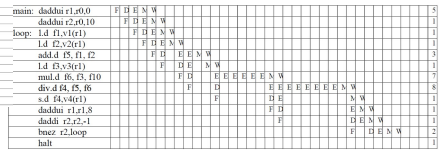
final content	
0	
1	
2	1
3	
4	1
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

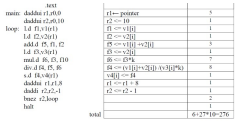
mis-predicted	6

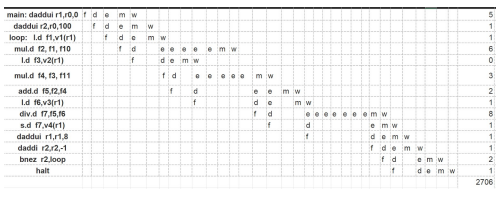


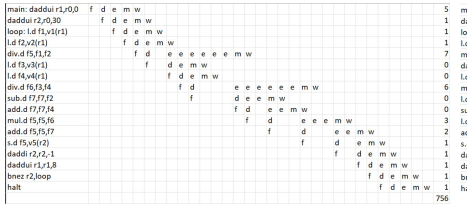


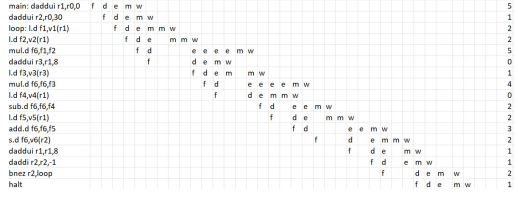
- load k bad Mem k end hony k wait
- stall k neche stall
- store instr mei load wali rangeen mazaji nai
- branch decision at decode stage so 1 stall after F
- halt in loop











iteration	issue	exe	mem	cdb	commit
1 l.d f1,v1(r1)	1	2	3	4	5
1 l.d f2,v2(r1)	1	3	4	5	6
1 l.d f3,v3(r1)	2	4	5	6	7
1 div.d f5,f3,f11	2	7	x	15	16
1 sub.d f4,f1,f2	3	6	x	8	17
1 add.d f6,f4,f5	3	16	x	18	19
1 div.d f7,f6,f12	4	23	x	31	32
1 s.d f7,v4(r1)	4	5	x	x	33
1 daddui r1,r1,8	5	6	x	7	34
1 daddi r2,r2,-1	5	7	x	9	35
1 bnez r2,loop	6	10	x	x	36
2 l.d f1,v1(r1)	7	8	9	10	37
2 l.d f2,v2(r1)	7	9	10	11	38
2 l.d f3,v3(r1)	8	10	11	12	39
2 div.d f5,f3,f11	8	15	x	23	40
2 sub.d f4,f1,f2	9	12	x	14	41
2 add.d f6,f4,f5	9	24	x	26	42
2 div.d f7,f6,f12	10	31	x	39	43
2 s.d f7,v4(r1)	10	11	x	x	44
2 daddui r1,r1,8	11	12	x	13	45
2 daddi r2,r2,-1	11	13	x	16	46
2 bnez r2,loop	12		x	x	47

First name, Last name, ID.....

c 63 o 75 d 64 i 69

Address		Code	BHT	Iteration #1		Iteration #2		Iteration #3		Iteration #4	
			entry #	prediction	result	prediction	result	prediction	result	prediction	result
0x0000		daddui r1, r0, 2									
0x0004		daddi r2, r0, -1									
0x0008		dadduı r5, r0, 0									
0x000c		dadduı r6, r0, 0									
0x0010	cyc:	dadduı r2, r2, 1									
0x0014		lb r3, vec(r2)									
0x0018		beqz r3, term	6	NT	NT	NT	NT	NT	NT	NT	NT
0x001c		dadduı r4, r3, r0									
0x0020		ddivu r3, r3, r1									
0x0024		dmulu r3, r3, r1									
0x0028		dsubu r4, r4, r3									
0x002c		bnez r4, nxt	3	NT	T	T	T	T	NT	NT	T
0x0030		dadduı r5, r5, 1									
0x0034		j cyc									
0x0038	nxt:	beqz r4, cyc	6	NT	NT	NT	NT			NT	NT
0x003c		dadduı r6, r6, 1									
0x0040		j cyc									
0x0044	term:	sb r5, reset(r0)									
0x0048		sb r6, reso(r0)									
0x004c	halt										

n 74 o 67

Address		Code	BHT	Iteration #5		Iteration #6		Iteration #7		Iteration #8	
			entry #	prediction	result	prediction	result	prediction	result	prediction	result
0x0000		dadduı r1, r0, 2									
0x0004		daddi r2, r0, -1									
0x0008		dadduı r5, r0, 0									
0x000c		dadduı r6, r0, 0									
0x0010	cyc:	dadduı r2, r2, 1									
0x0014		lb r3, vec(r2)									
0x0018		beqz r3, term	6	NT	NT	NT	NT	NT	T		
0x001c		dadduı r4, r3, r0									
0x0020		ddivu r3, r3, r1									
0x0024		dmulu r3, r3, r1									
0x0028		dsubu r4, r4, r3									
0x002c		bnez r4, nxt	3	T	NT	NT	T				
0x0030		dadduı r5, r5, 1									
0x0034		j cyc									
0x0038	nxt:	beqz r4, cyc	6			NT	NT				
0x003c		dadduı r6, r6, 1									
0x0040		j cyc									
0x0044	term:	sb r5, reset(r0)									
0x0048		sb r6, reso(r0)									
0x004c	halt										

The number of mispredicted branches during the execution of the code is: ___6___

BHT - Final content

Entry 0	0	Entry 4	0
Entry 1	0	Entry 5	0
Entry 2	0	Entry 6	1
Entry 3	1	Entry 7	0

0 2 2 1

Address	Code	BHT	Iteration #1		Iteration #2		Iteration #3		Iteration #4	
		entry #	prediction	result	prediction	result	prediction	result	prediction	result
0x0000	dadduı r1, r0, 1									
0x0004	dadduı r2, r0, 0									
0x0008	dadduı r3, r0, 0									
0x000c	dadduı r4, r0, 0									
0x0010	dadduı r5, r0, 6									
0x0014	cyc: beq r2, r5, term	5	NT	NT	NT	NT	NT	NT	NT	NT
0x0018	lb r6, vec(r2)									
0x001c	beq r6, r0, eq0	7	NT	T	T	NT	NT	NT	NT	NT
0x0020	beq r6, r1, eq1	8			NT	NT	NT	NT	NT	T
0x0024	j nxt									
0x0028	eq0: dadduı r3, r3, 1									
0x002c	j nxt									
0x0030	eq1: dadduı r4, r4, 1									
0x0034	nxt: dadduı r2, r2, 1									
0x0038	j cyc									
0x003c	term: sb r3, res0(r2)									
0x0040	sb r4, res1(r2)									
0x0044	halt									

0 0 1

Address	Code	BHT	Iteration #5		Iteration #6		Iteration #7		Iteration #8	
		entry #	prediction	result	prediction	result	prediction	result	prediction	result
0x0000	dadduı r1, r0, 1									
0x0004	dadduı r2, r0, 0									
0x0008	dadduı r3, r0, 0									
0x000c	dadduı r4, r0, 0									
0x0010	dadduı r5, r0, 6									
0x0014	cyc: beq r2, r5, term	5	NT	NT	NT	NT	NT	NT	NT	T
0x0018	lb r6, vec(r2)									
0x001c	beq r6, r0, eq0	7	NT	T	T	T	T	NT		
0x0020	beq r6, r1, eq1	8					T	T		
0x0024	j nxt									
0x0028	eq0: dadduı r3, r3, 1									
0x002c	j nxt									
0x0030	eq1: dadduı r4, r4, 1									
0x0034	nxt: dadduı r2, r2, 1									
0x0038	j cyc									
0x003c	term: sb r3, res0(r2)									
0x0040	sb r4, res1(r2)									
0x0044	halt									

The number of mispredicted branches during the execution of the code is: ___6___

BHT - Final content

Entry 0	0	Entry 4	0	Entry 8	1	Entry 12	0
Entry 1	0	Entry 5	1	Entry 9	0	Entry 13	0
Entry 2	0	Entry 6	0	Entry 10	0	Entry 14	0
Entry 3	0	Entry 7	0	Entry 11	0	Entry 15	0