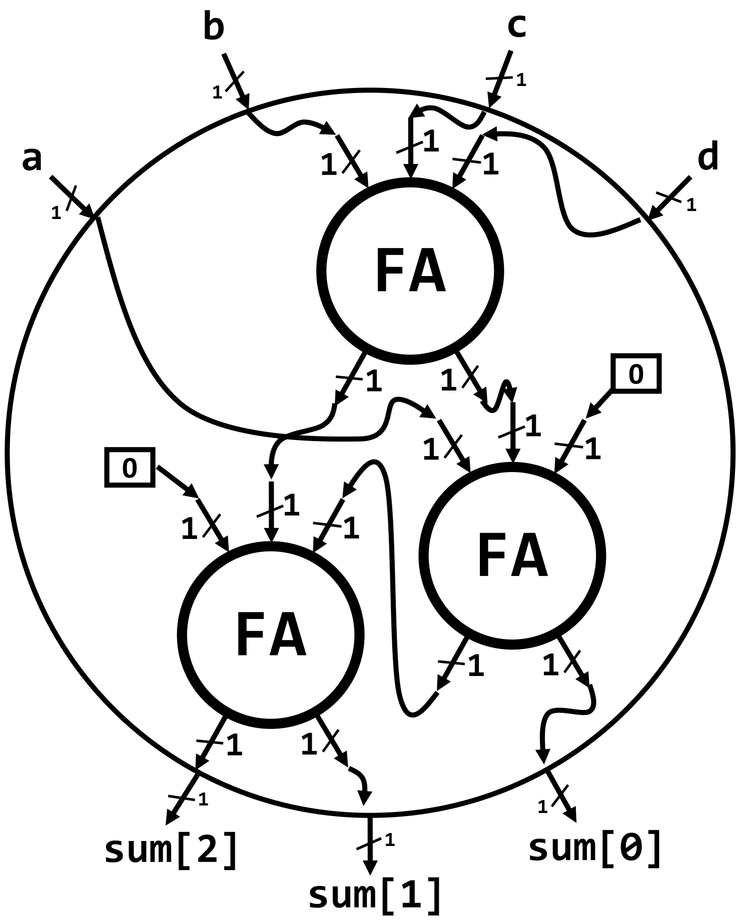
(d) [5 points] Interestingly, the full-adder can also be used to add four 1-bit input tokens. This is a natural extension of the full-adder in the same way we extended the half-adder to create the full-adder itself (in part (b)). Implement the 4-input node below using only a *minimum* number of full-adders (FA) (i.e., the dataflow node you designed in part (b)). *Hint: you may use constant input tokens if necessary.*



(e) [15 points] As it turns out, any $n \geq 3$ 1-bit input binary adders can be implemented purely using full-adders. Fill in the table below for the *minimum* number of required full adders to implement an $n$-input 1-bit adder.

| $n$ | **# required full-adders** |
|---|---|
| 3 | 1 |
| 4 | 3 |
| 5 | 3 |
| 6 | 4 |
| 7 | 4 |
| 8 | 7 |

## 12 BONUS: Branch Prediction [40 points]

Assume a processor that implements an ISA with eight registers (R0-R7). In this ISA, the main memory is byte-addressable and each word contains 4 bytes. The processor employs a branch predictor. The ISA implements the instructions given in the following table:

| **Instructions** | **Description** |
|---|---|
| la $R_i$, Address | load the *Address* into $R_i$ |
| move $R_i$, $R_j$ | $R_i \leftarrow R_j$ |
| move $R_i$, $(R_j)$ | $R_i \leftarrow$ Memory$[R_j]$ |
| move $(R_i)$, $R_j$ | Memory$[R_i] \leftarrow R_j$ |
| li $R_i$, Imm | $R_i \leftarrow$ Imm |
| add $R_i$, $R_j$, $R_k$ | $R_i \leftarrow R_j + R_k$ |
| addi $R_i$, $R_j$, Imm | $R_i \leftarrow R_j +$ Imm |
| cmp $R_i$, $R_j$ | Compare: Set sign flag, if $R_i < R_j$; set zero flag, if $R_i = R_j$ |
| cmp $R_i$, $(R_j)$ | Compare: Set sign flag, if $R_i <$ Memory$[R_j]$; set zero flag, if $R_i =$ Memory$[R_j]$ |
| cmpi $R_i$, Imm | Compare: Set sign flag, if $R_i <$ Imm; set zero flag, if $R_i =$ Imm. |
| jg label | Jump to the target address if **both** of sign and zero flags are zero. |
| jnz label | Jump to the target address if zero flag is zero. |
| halt | Stop executing instructions. |

The processor executes the following program. Answer the questions below related to the accuracy of the branch predictors that the processor can potentially implement.

```
       la R0, Array
       move R6, R0
       li R1, 4
       move R5, R1
       move R7, R1
       move R2, R0
       addi R2, R2, 4
Loop:
       move R3, (R2)
       cmp R3, (R0)
       jg Next_Iteration
       move R4, (R0)
       move (R0), R3
       move (R2), R4
Next_Iteration:
       addi R0, R0, 4
       addi R2, R2, 4
       addi R1, R1, -1
       cmpi R1, 0
       jnz Loop
       move R1, R7
       addi R5, R5, -1
       move R0, R6
       move R2, R0
       addi R2, R2, 4
       cmpi R5, 0
       jnz Loop
       halt
.data
Array: word 5, 20, 1, -5, 34
```

(a) [15 points] What would be the prediction accuracy using a global one-bit-history (last-time) branch predictor shared between *all* the branches? The initial state of the predictor is "taken".

**Answer:** 19/36.

Note that initial values of both $R_1$ and $R_5$ are 4; and they change only before the branches in lines 20 and 27 respectively. Both branches follow the pattern of T-T-T-NT, which creates a nested loop.

At each iteration of the internal loop, adjacent elements (pointed by $R_0$ and $R_2$) are swapped, if $Memory[R_0] \leq Memory[R_2]$. Then, both $R_0$ and $R_4$ are incremented by 4. So they point to the next element in the next iteration.

Therefore, the code sorts the elements in *Array* in increasing order.

Table below shows the behavior of each branch through the code. Here T means that the corresponding branch is taken at specified turn, whereas N indicates that it is not taken.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Line11 | T | | N | | N | | T | | | N | | N | | T | | T | | | N | | T | | T | | T | | | T | | T | | T | | T | | |
| Line20 | | T | | T | | T | | N | | | T | | T | | T | | N | | | T | | T | | T | | N | | | T | | T | | T | | N | |
| Line27 | | | | | | | | | T | | | | | | | | | T | | | | | | | | | T | | | | | | | | | N |

One-bit-history branch predictor suggests that the next branch's behavior will be the same with the last one. Table below shows the predictor states, hits, and misses through the execution.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Predictor State | T | T | T | N | T | N | T | T | N | T | N | T | N | T | T | T | T | N | T | N | T | T | T | T | T | T | N | T | T | T | T | T | T | T | T | N |
| Branch Behavior | T | T | N | T | N | T | T | N | T | N | T | N | T | T | T | T | N | T | N | T | T | T | T | T | T | N | T | T | T | T | T | T | T | T | N | N |
| Hit/Miss | H | H | M | M | M | M | H | M | M | M | M | M | M | H | H | H | M | M | M | M | H | H | H | H | H | M | M | H | H | H | H | H | H | H | M | H |

(b) [15 points] What would be the prediction accuracy using a global two-bit-history (two-bit counter) branch predictor shared between *all* the branches? Assume that the initial state of the two-bit counter is "weakly taken". The "weakly taken" state transitions to the "weakly not-taken" state on misprediction. Similarly, the "weakly not-taken" state transitions to the "weakly taken" state on misprediction. A correct prediction in one of the "weak" states transitions the state to the corresponding "strong" state.

**Answer:** 26/36.

**Explanation:**
Table below shows the predictor states, hits, and misses through the code. Used abbreviations are as follows: ST: Strongly Taken, WT: Weakly Taken, WN: Weakly Not-taken, SN: Strongly Not-taken.

Branch behavior is the same with question (a), since both of them are shared predictors.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Predictor State | WT | ST | ST | WT | ST | WT | ST | ST | WT | ST | WT | ST | WT | ST |
| Branch Behavior | T | T | N | T | N | T | T | N | T | N | T | N | T | T |
| Hit/Miss | H | H | M | H | M | H | H | M | H | M | H | M | H | H |

| | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Predictor State | ST | ST | ST | WT | ST | WT | ST | ST | ST | ST | ST | ST | WT | ST |
| Branch Behavior | T | T | N | T | N | T | T | T | T | T | T | N | T | T |
| Hit/Miss | H | H | M | H | M | H | H | H | H | H | H | M | H | H |

| | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 |
|---|---|---|---|---|---|---|---|---|
| Predictor State | ST | ST | ST | ST | ST | ST | ST | WT |
| Branch Behavior | T | T | T | T | T | T | N | N |
| Hit/Miss | H | H | H | H | H | H | M | M |