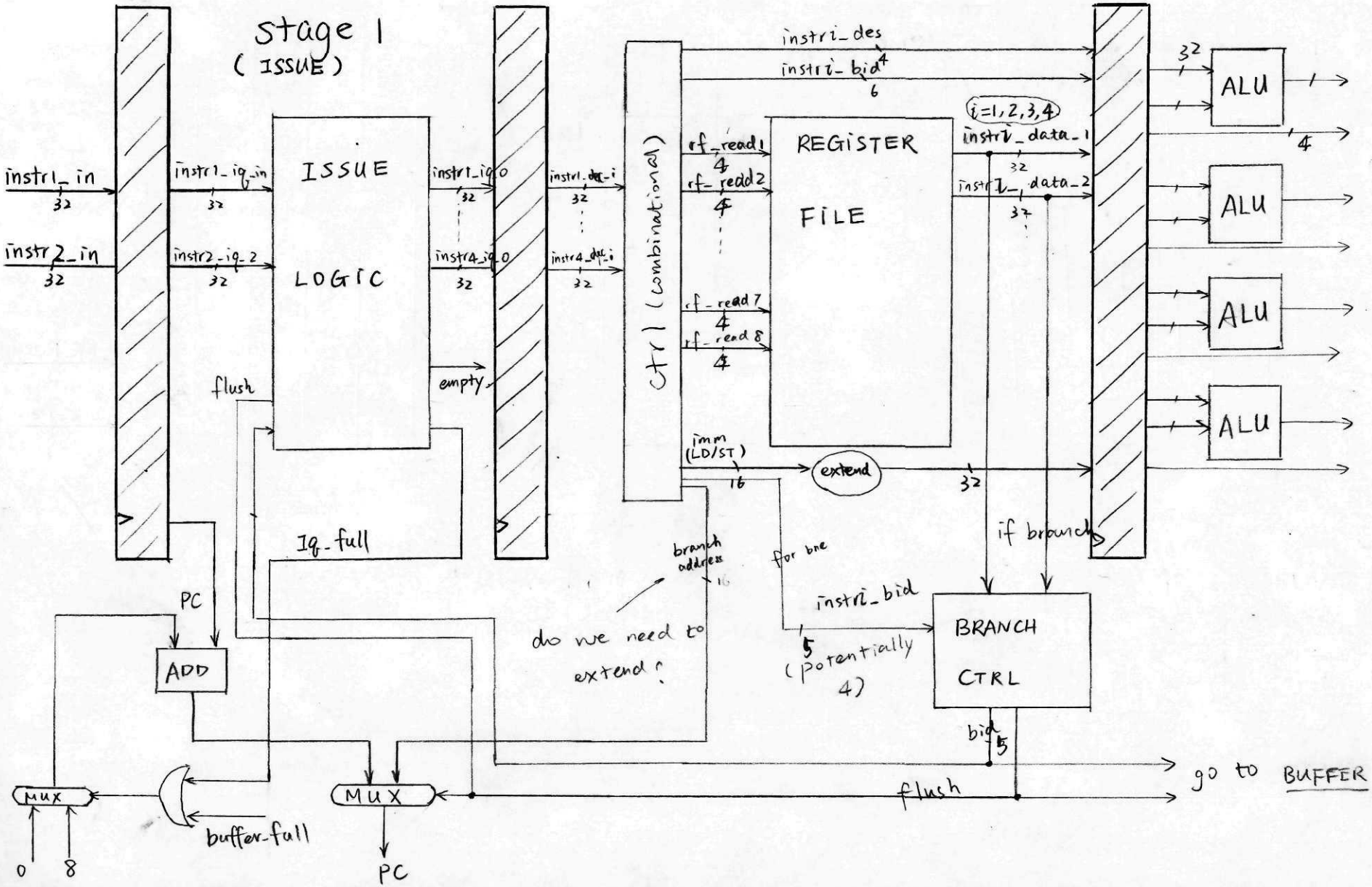
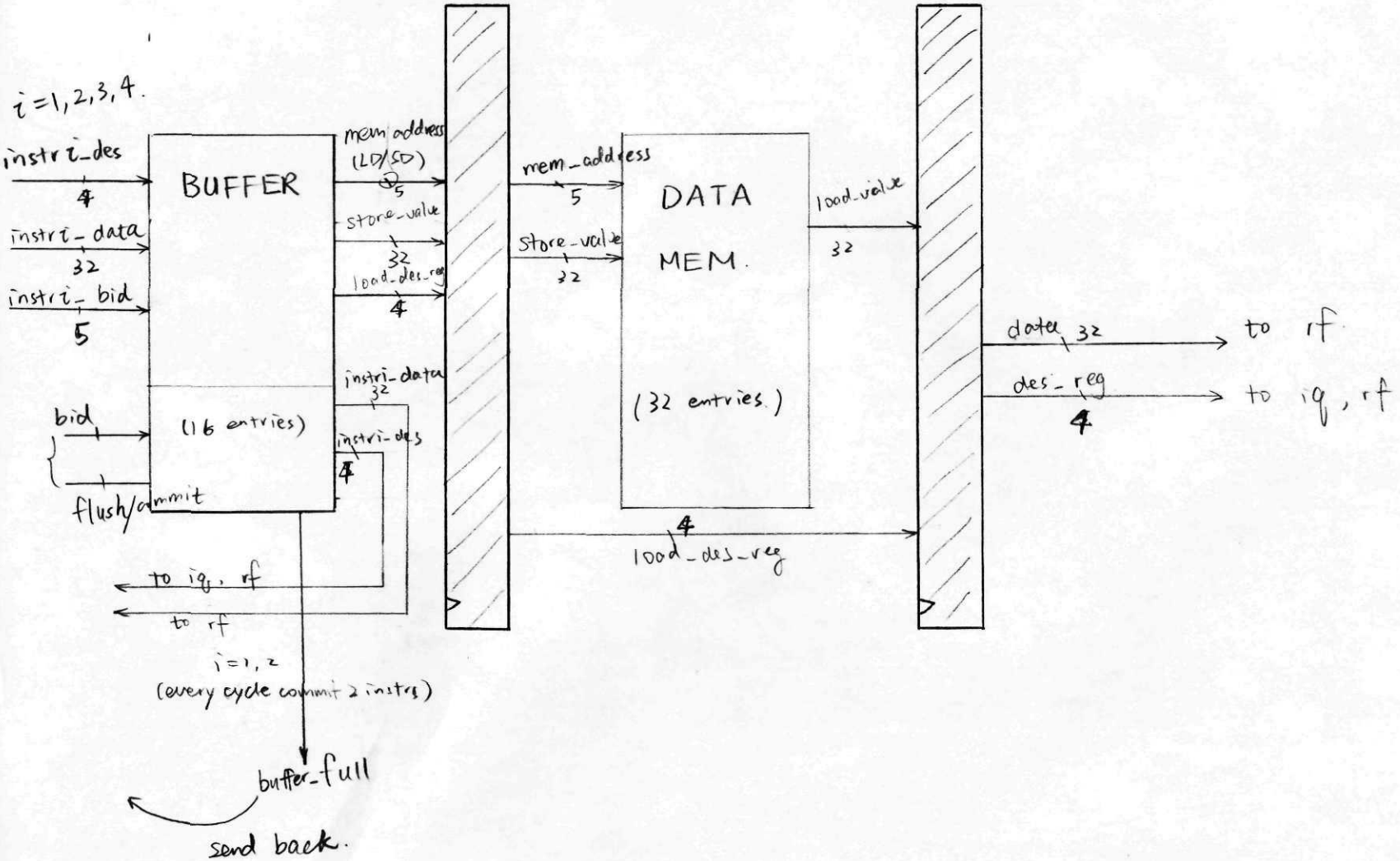

stage 1
(ISSUE)
instr1_in
32
instr2_in
32
instr1_iq_in
32
instr2_iq_2
32
ISSUE
LOGIC
instr1_iq_0
32
instr4_iq_0
32
flush
empty
Iq_full
instr1_dst_i
32
instr4_dst_i
32
ctrl (combinational)
instri_des
instri_bid
4
6
rf_read1
4
rf_read2
4
rf_read7
4
rf_read8
4
REGISTER
FILE
i=1,2,3,4
instri_data_1
32
instri_data_2
32
imm
(LD/ST)
16
extend
32
if branch
32
ALU
4
ALU
ALU
ALU
PC
ADD
branch address
16
do we need to extend?
for bne
instri_bid
5
(potentially 4)
BRANCH
CTRL
bid
5
MUX
MUX
buffer_full
0
8
PC
flush
go to BUFFER


i = 1, 2, 3, 4.
instr i_des
4
instr i_data
32
instr i_bid
5
BUFFER
(16 entries)
bid
flush/commit
mem address (LD/SD)
5
store_value
32
load_des_reg
4
instr i_data
32
instr i_des
4
to iq, rf
to rf
i = 1, 2
(every cycle commit 2 instrs)
buffer_full
send back.
mem_address
5
store_value
32
DATA MEM.
(32 entries.)
load_value
32
4
load_des_reg
data, 32
to rf
des_reg
4
to iq, rf

Stage 1 (ISSUE) (1)

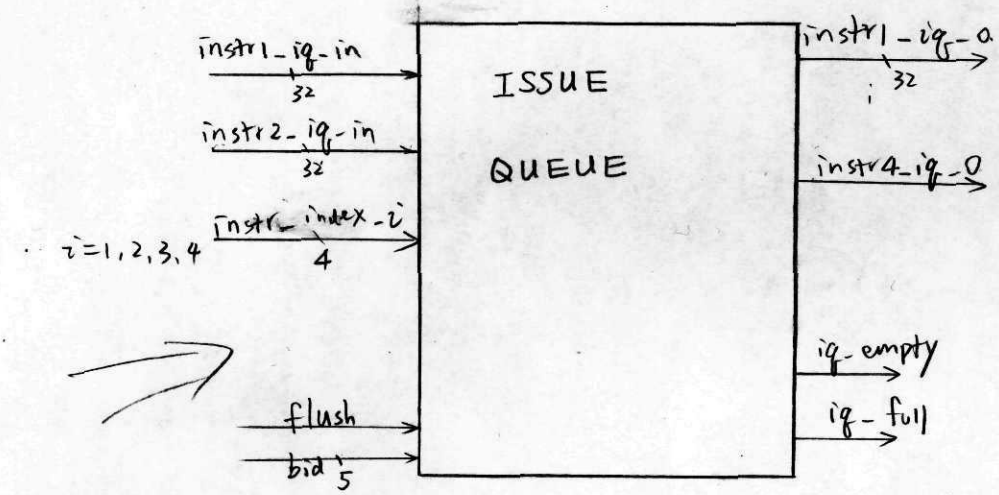
1. Basic Functionality

   ① Issue queue

   ② Hazard check

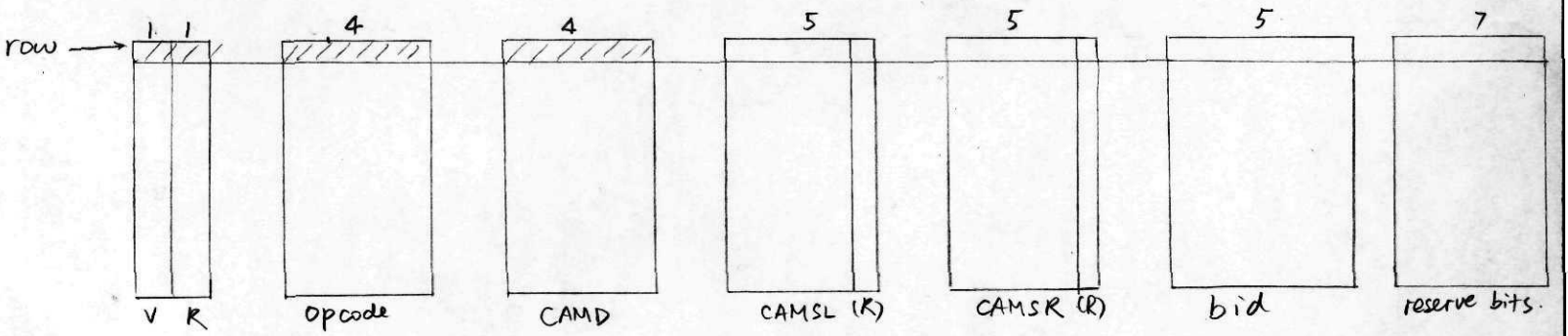
2. Basic Blocks

   a. Issue queue



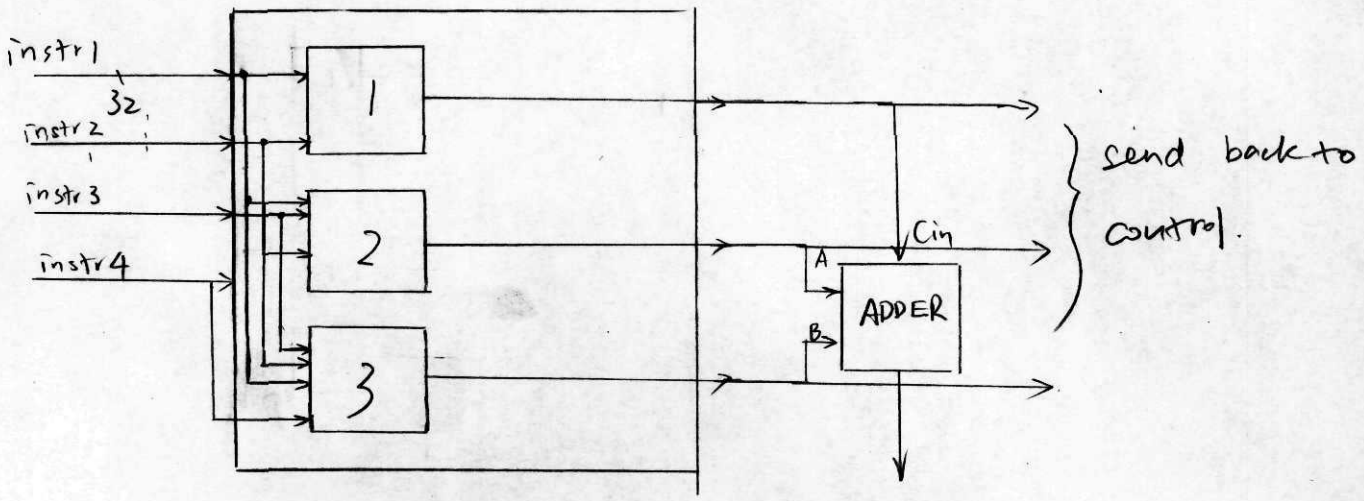
control signal

The issue queue has 16 entries. It contains several parts.

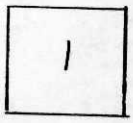


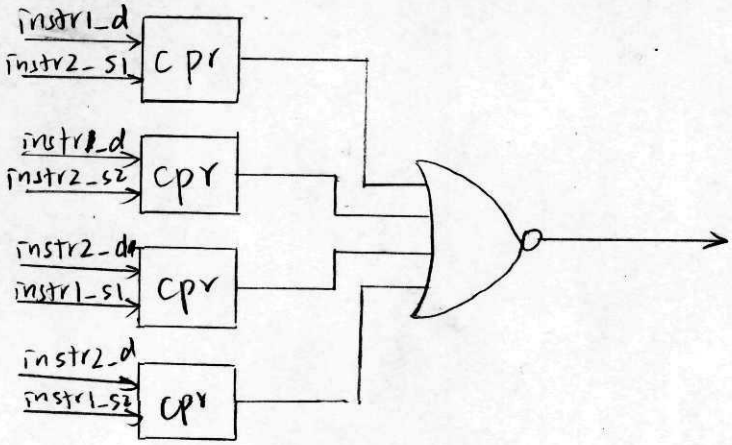
Stage 1 (2)

b. Hazard Checker



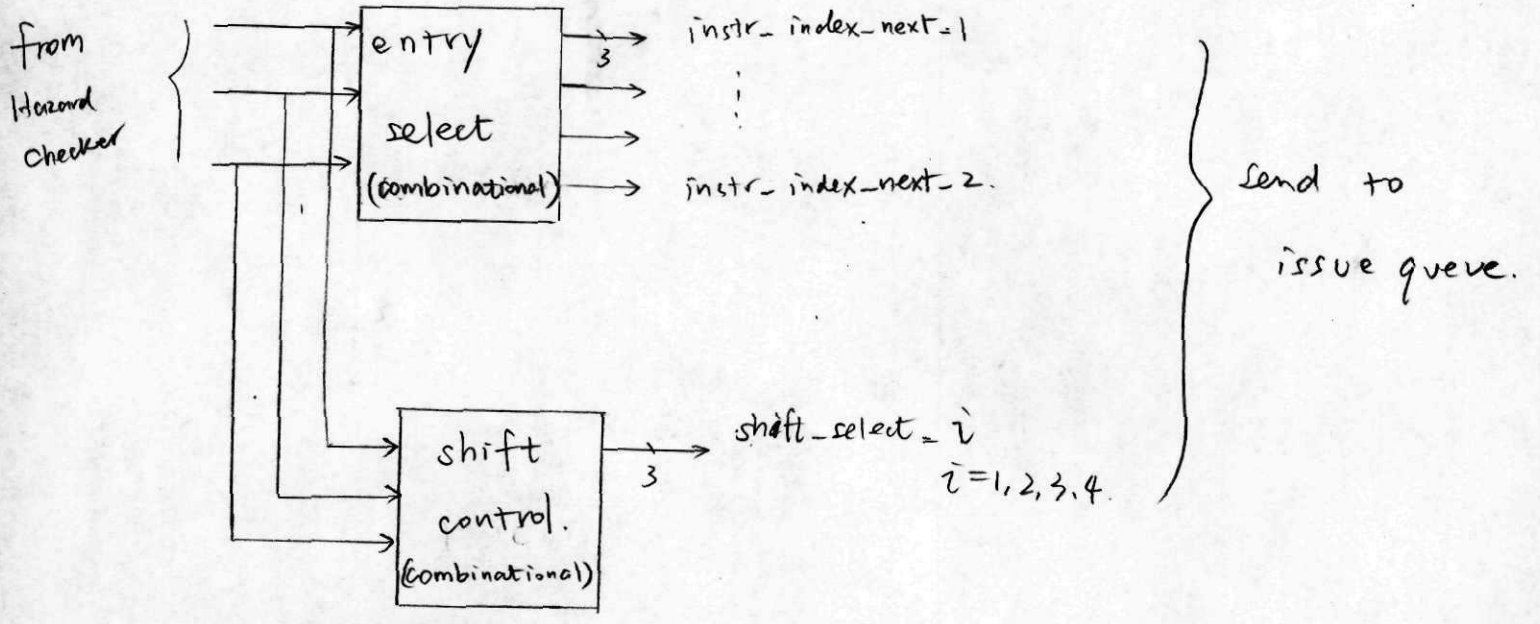
in





Stage 1 (3)

c. control logic



Truth Table:

Entry Select

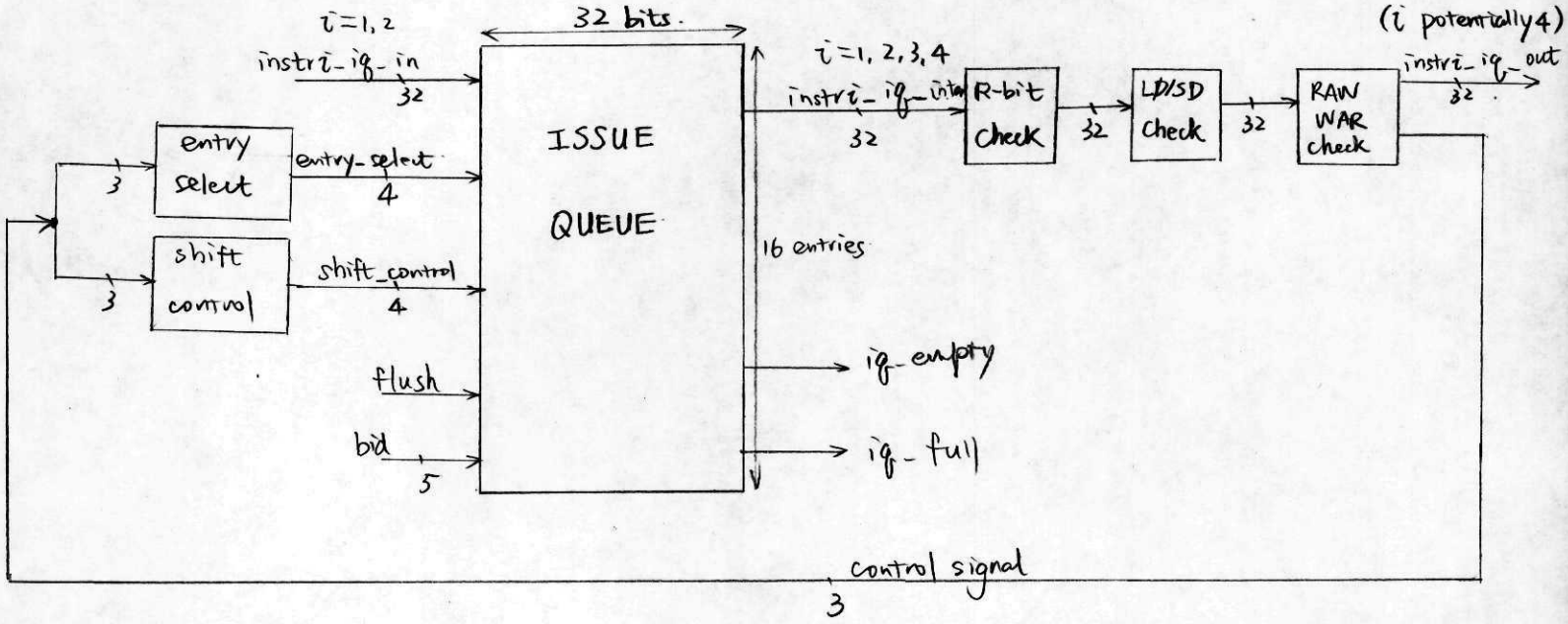
3-bit feedback	selected entries.
0 0 0	1, 2, 3, 4
0 0 1	2, 3, 4, 5
0 1 0	1, 3, 4, 5
0 1 1	3, 4, 5, 6
1 0 0	1, 2, 4, 5
1 0 1	2, 4, 5, 6
1 1 0	1, 4, 5, 6
1 1 1	4, 5, 6, 7.

Stage 1 (4)

Shift Control

entry	1	2	3	rest
0	1	1	1	1
1	0	2	2	2
2	1	0	2	2
3	0	0	3	3
4	1	0	0	3
5	0	2	0	3
6	1	0	0	3
7	0	0	0	4

d. Put it all together.



Stage 2 DECODE (1)

1. Basic Functionality

   ① Decode

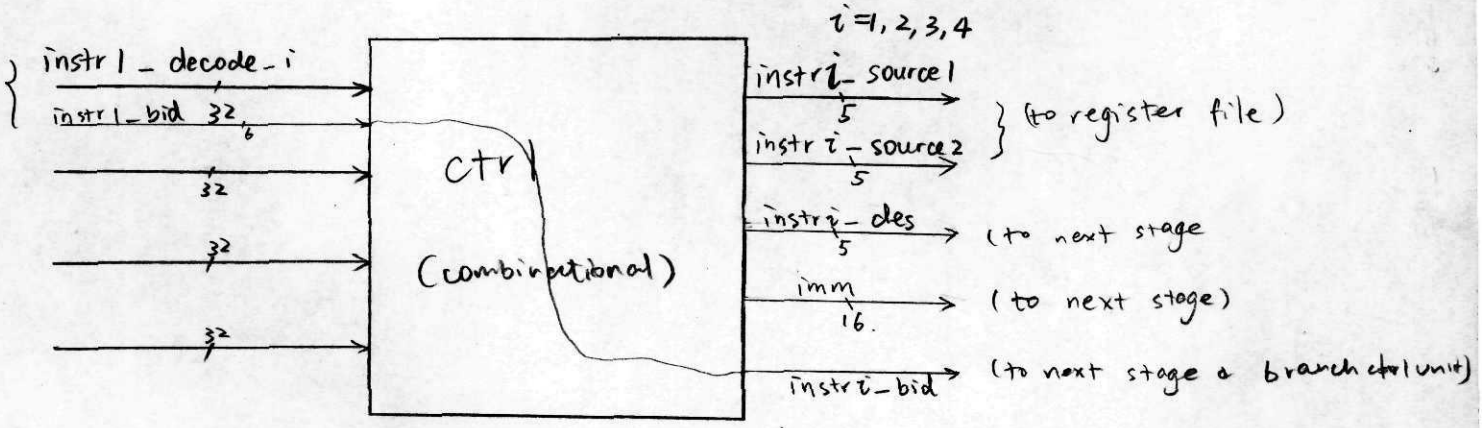
   ② Access Register File

   ③ Branch control

2. Basic Blocks.

   ① Ctrl. (combinational)

   a. input and output interfaces



   b. basic logic

   if (ADD) i=1,2,3,4

   instr i_decode_i (16~20) → instr i_source1

   instr i_decode_i (0~5) → instr i_source2.

   instr i_decode_i (21~25) → instr i_des

   if (BNE)

   instr i_decode_i (16~20) → instr i_source1

   instr i_decode_ (21~25) → instr i_source2

   x x x → instr i_des

Stage 2 DECODE (2)

if (LD)

instri_decode_i(16~20) → instri_source1.

× × × → instri_source2.

instri_decode_i(21~25) → instri_des.

if (SD)

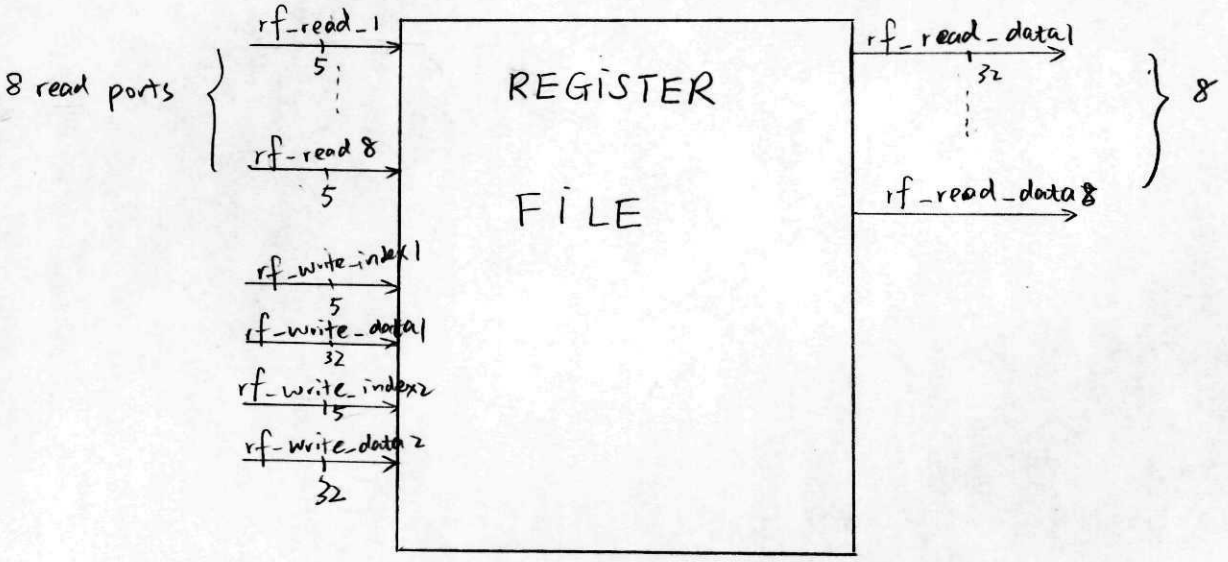
instri_decode_i(16~20) → instri_source1

instri_decode_i(21~25) → instri_source2

×××× → instri_des.

② Register File

a. input & output interfaces

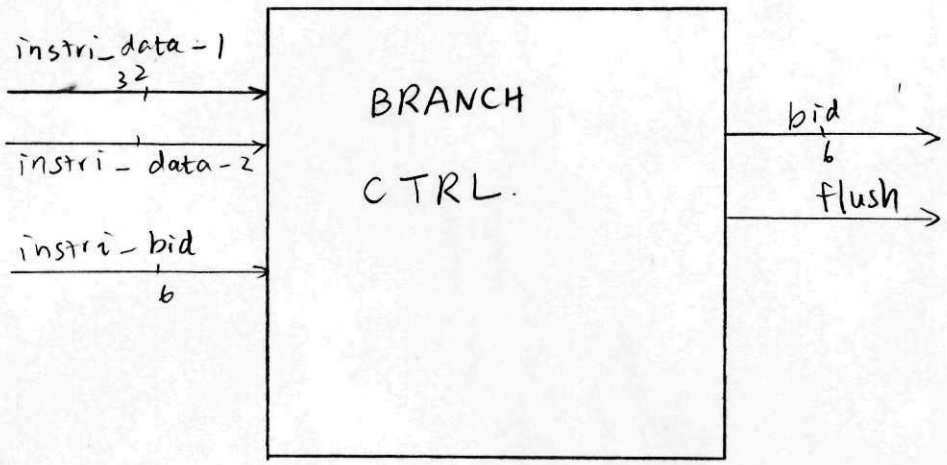


16 entries.

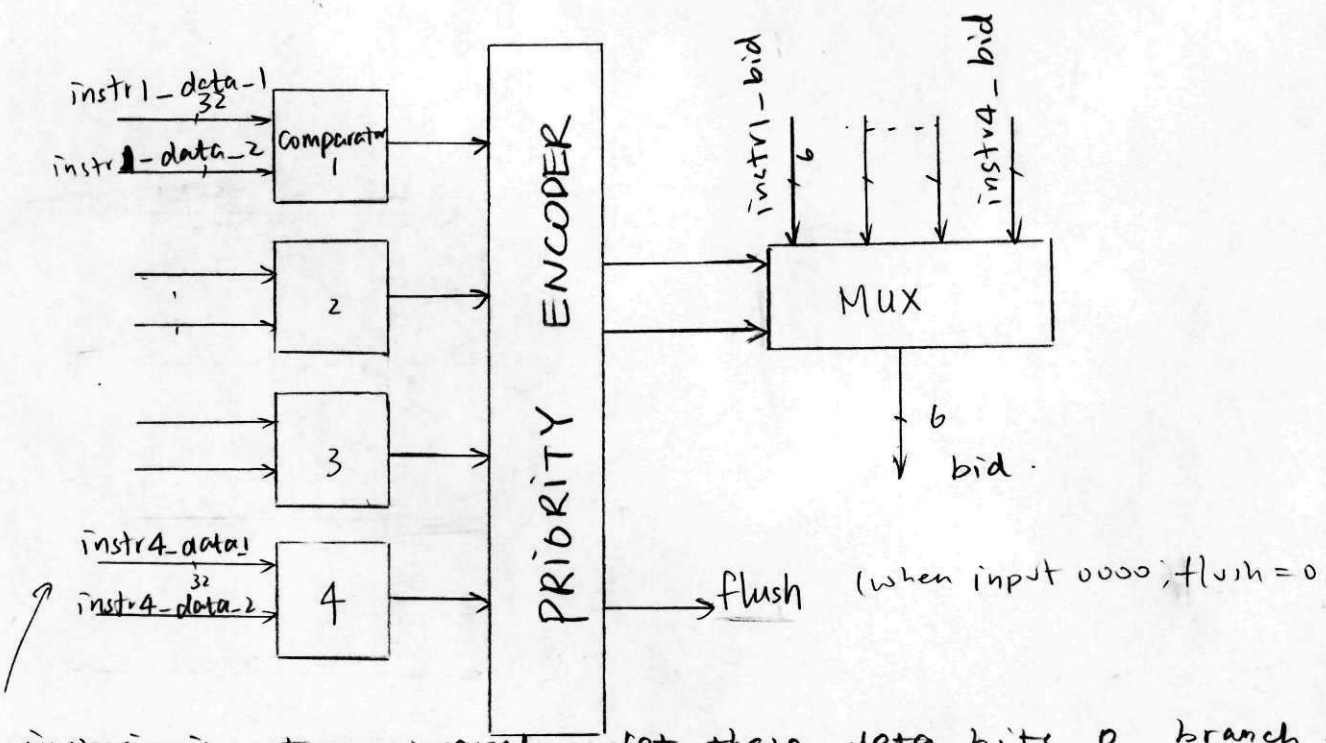
Stage 2 DECODE (3/4)

③ branch control unit.

a. input and output interfaces.



b. microarchitecture



① if instr1 is not a branch, set these data bits 0. branch not taken, predict correctly.

② if equal, output 0.

③ if more than 1 branches mispredict, use priority encoder to choose the first mispredicted branch, set flush signal and flush all the instructions after that branch.

Stage 3 EXECUTION

1. Basic Functionality.

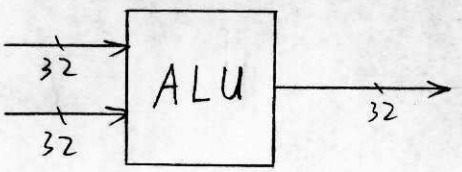
   ① For LD/SD instructions, add the imm and the source to get the memory address

   ② For ADD instructions, add the data from two source registers to get the result for the destination register.

   ③ To speculate the branch, store the data, corresponding destination and branch id. After a branch is resolved, if it predicted correctly, the data can be committed, otherwise these data should be flushed.

2. Basic Blocks.

   ① ALU



Stage 3 (2)

② Buffer.

a. input & output interfaces.