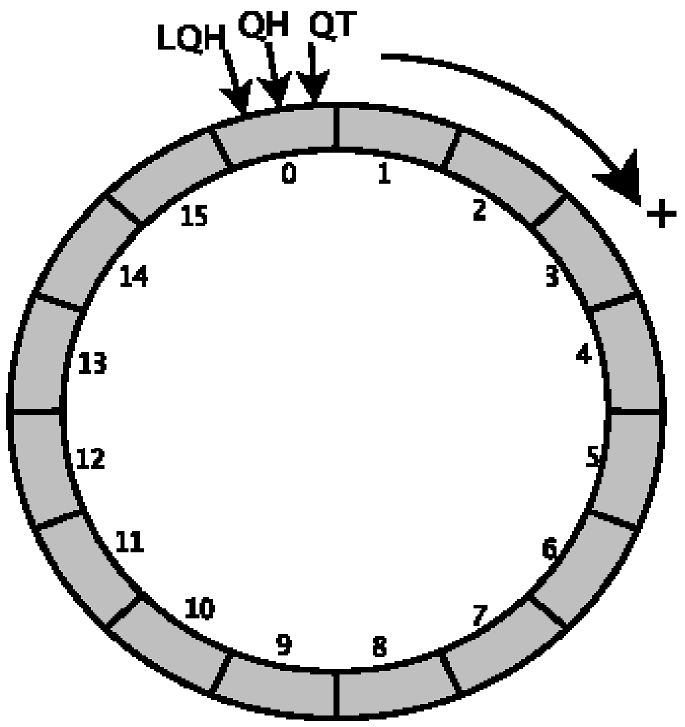
QueueCore – The Strong Wave!

Dr. Abderazek Ben Abdallah

May 2007

The University of Electro-communications
Graduate School of Information Systems



Background & Introduction (1/3)

- FIFO for temporary results have drawn the attention of computer architects.
- In 1981 the concept of a simple machine based on Queue was reported.
- University of Edinburgh in 1998 investigated the usage of queue register file for VLIW machines.

Background & Introduction (2/3)

- Carnegie Melon University reported in 2002 the usage of a simple queue machine for dynamic compilation of SW to HW

- No one of these researches investigated real model or architecture based on Queue

- In his Ph.D. (1999), Abderazek proposed a novel Queue processor model based on circular Queue-register.

Background & Introduction (3/3)

- I present a model, architecture and design of a novel Produced order Queue processor (QueueCore)

- **The Queue model has**:
  - High natural ILP ← Grouped ILP
  - Simple hardware ← no aggressive hardware techniques
  - Implicit references → reduced-bit instruction → small program size
  - Single assignment rule → no false dependencies → no register reaming

- **The QueueCore is targeted for**:
  - Applications constrained in:
    - ✓ Memory (16-bit on 32 data path)
    - ✓ Area (no RR, no aggressive ILP extraction)
    - ✓ Power consumption ← 1,2

First-In First-Out data structure (queue)

- Elements are inserted (enqueued) at the rear (tail) of the queue, or QT.
- Elements are taken (dequeued) at the head of the queue, or QH.

x = a+b

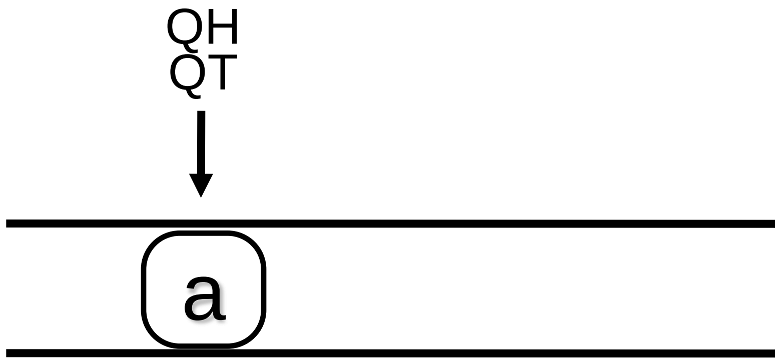
enqueue a

First-In First-Out data **structure** (queue)

- Elements are inserted (enqueued) at the rear (tail) of the queue, or QT.
- Elements are taken (dequeued) at the head of the queue, or QH.

x = a+b

enqueue a

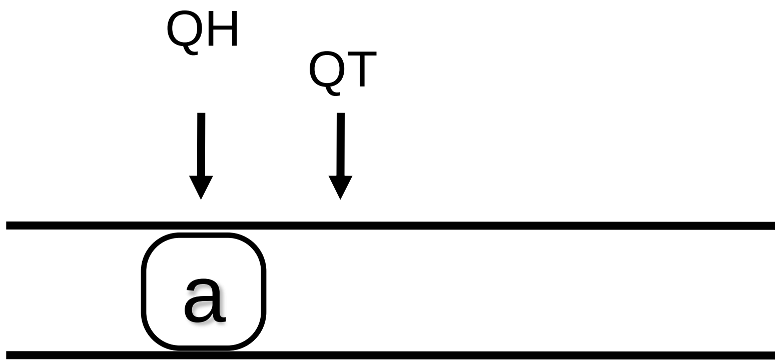


First-In First-Out data structure (queue)

- Elements are inserted (enqueued) at the rear (tail) of the queue, or QT.
- Elements are taken (dequeued) at the head of the queue, or QH.

x = a+b

enqueue a

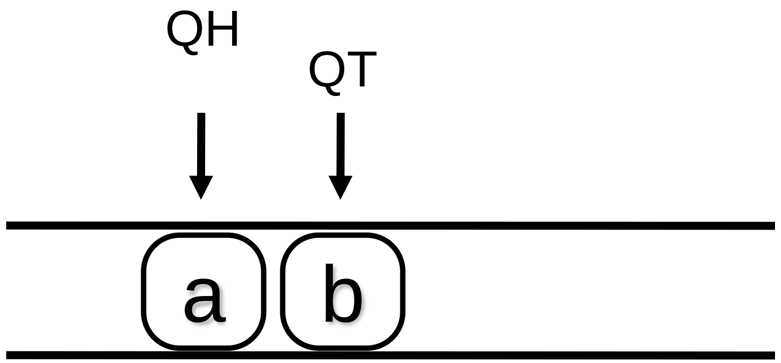


First-In First-Out data structure (queue)

- Elements are inserted (enqueued) at the rear (tail) of the queue, or QT.
- Elements are taken (dequeued) at the head of the queue, or QH.

x = a+b

enqueue a
enqueue b

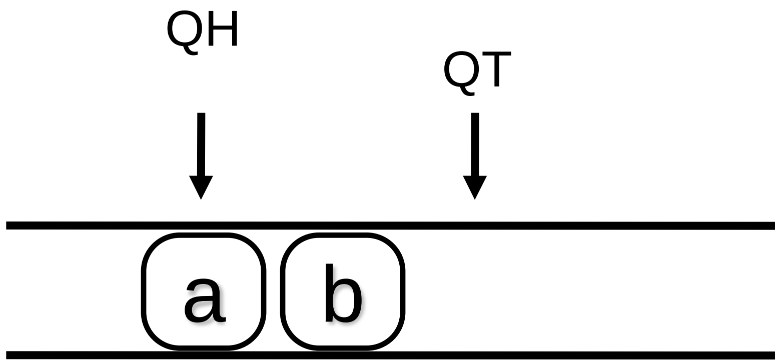


First-In First-Out data structure (queue)

- Elements are inserted (enqueued) at the rear (tail) of the queue, or QT.
- Elements are taken (dequeued) at the head of the queue, or QH.

x = a+b

enqueue a
enqueue b

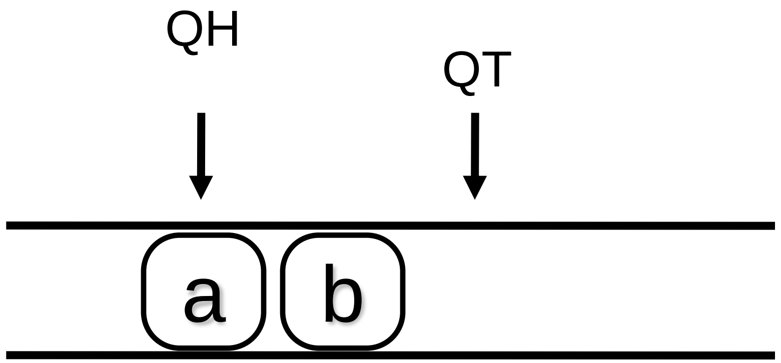


First-In First-Out data structure (queue)

- Elements are inserted (enqueued) at the rear (tail) of the queue, or QT.
- Elements are taken (dequeued) at the head of the queue, or QH.

x = a+b

enqueue a
enqueue b
add

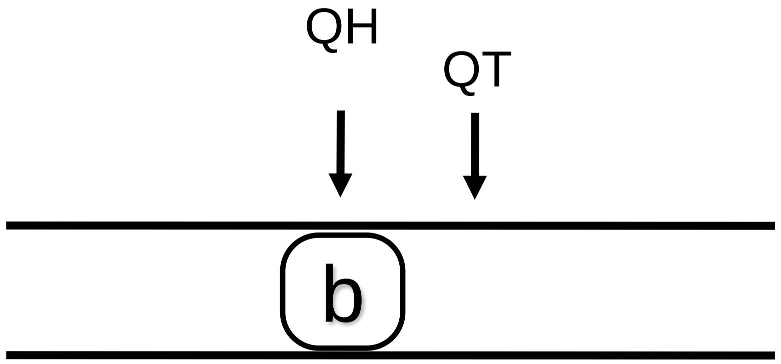


First-In First-Out data structure (queue)

- Elements are inserted (enqueued) at the rear (tail) of the queue, or QT.
- Elements are taken (dequeued) at the head of the queue, or QH.

x = a+b

enqueue a
enqueue b
add

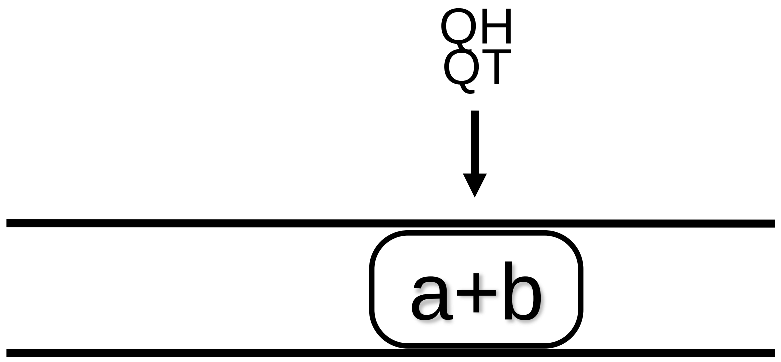


First-In First-Out data structure (queue)

- Elements are inserted (enqueued) at the rear (tail) of the queue, or QT.
- Elements are taken (dequeued) at the head of the queue, or QH.

x = a+b

enqueue a
enqueue b
add

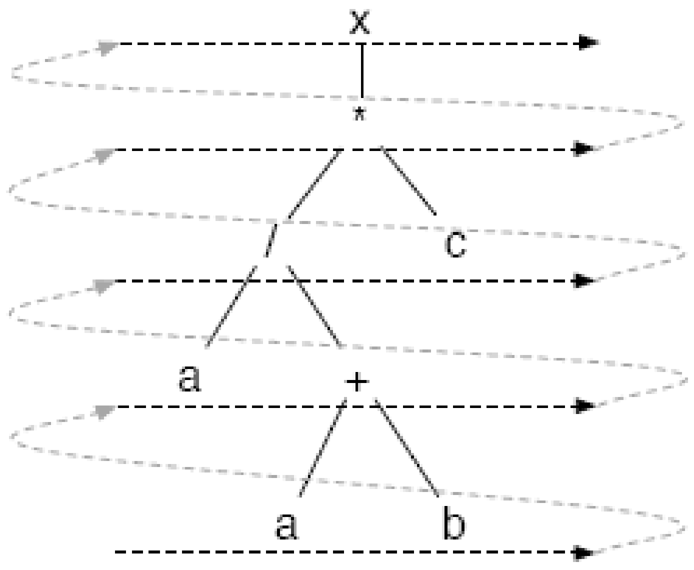


QEM Instruction Generation

- Queue Execution model instruction can be obtained by traversing a parse tree in LOST scheme.
- The arithmetic operations are the internal nodes
- The fetch operations are the leaf nodes
- We proved [1,2] that QEM can be used to evaluate any arbitrary tree
- QEM Inst. Sequence can be derived correctly from the parse three

1. B. A. Abderazek, Kirilka Nikolova, and M. Sowa. **On a Practical Queue Execution Model**, Proc. of the Int. Conf. on Circuits and Systems, Computers and Communications ICSCC01, pp.939-944, July 2001.
2. B. A. Abderazek, M. Sarem., and M. Sowa, **Acyclic DFG on a Queue Machine**, ICJSPP03, pp.119-120, 20003

Queue Model
Instructions generation



Level 4

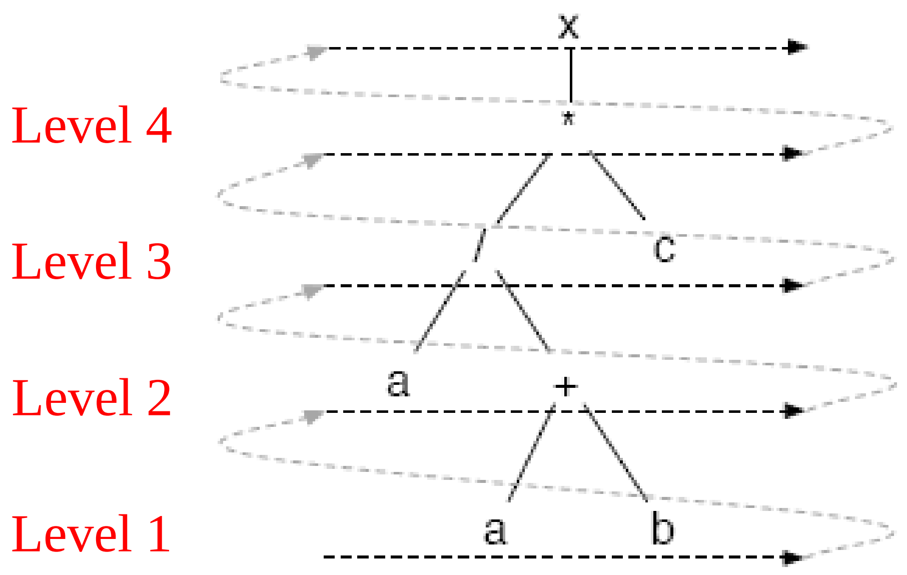
Level 3

Level 2

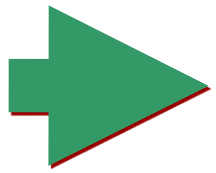
Level 1

a. Breadth-First Traversal

Queue Model
Instructions generation



a. Breadth-First Traversal

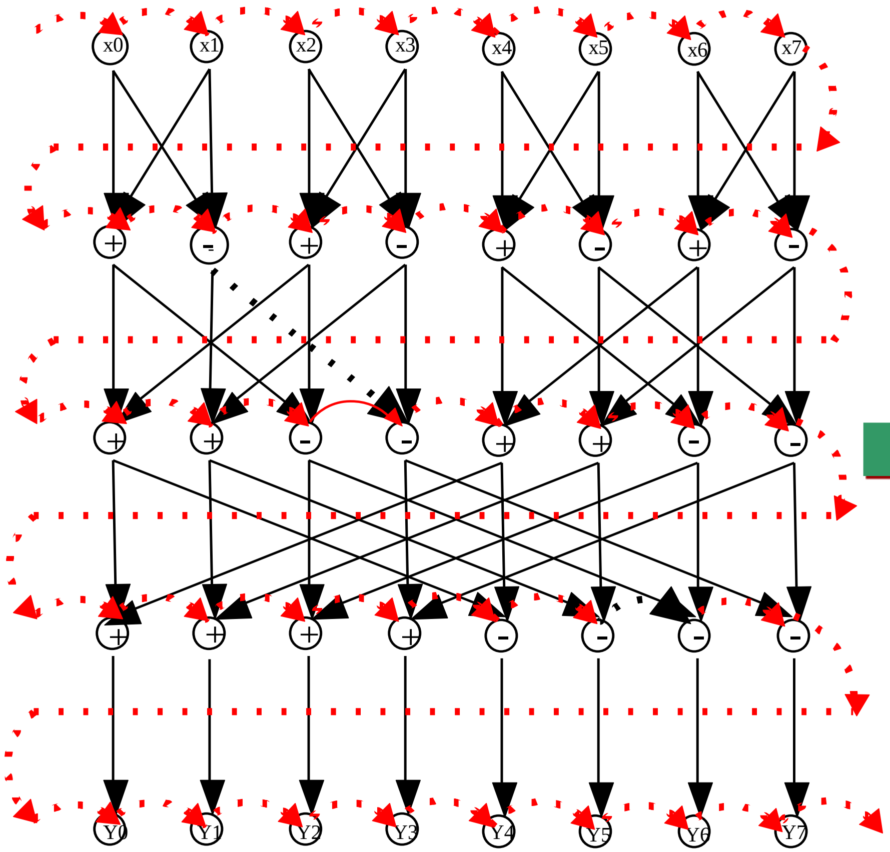


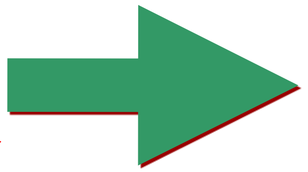
Operation	Queue Contents
enqueue a	a
enqueue b	a, b
enqueue a	a, b, a
add	a, a+b
div	a/(a+b)
enqueue c	a/(a+b), c
mul	[a/(a+b)]*c
dequeue x	ø

c. Evaluation in a queue

Queue Model

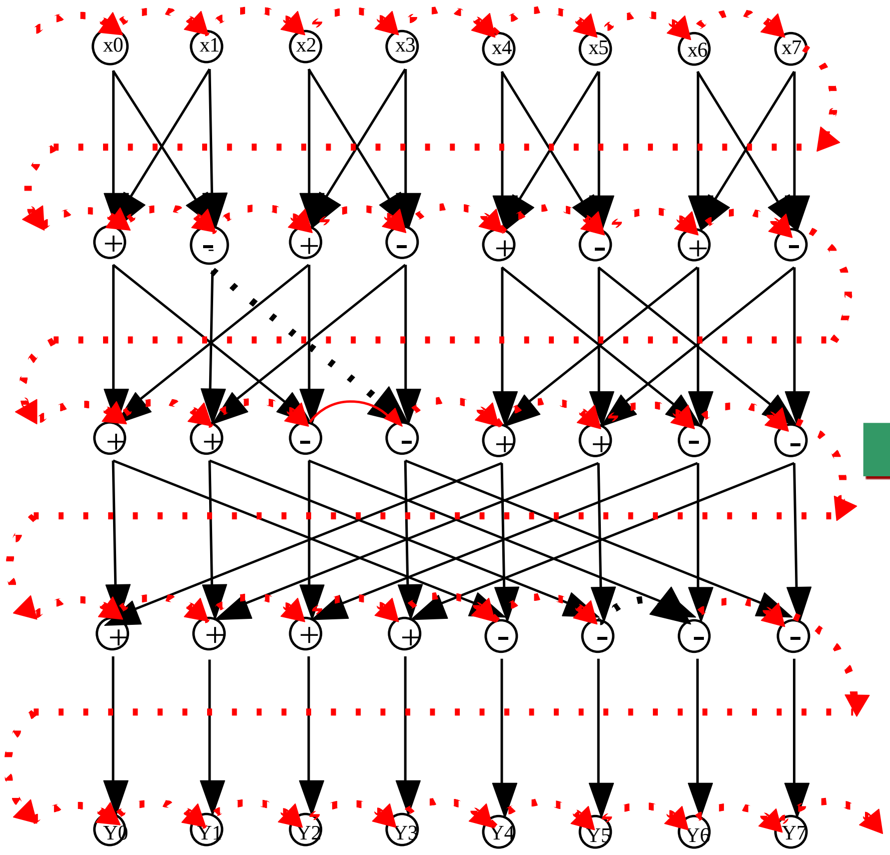
Instructions generation



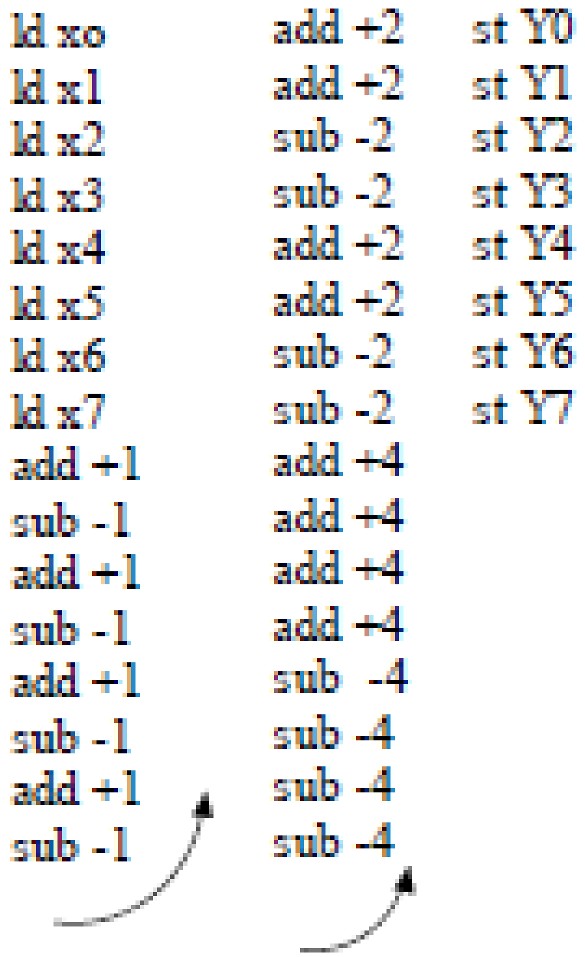


Breath first traversal

Queue Model
Instructions generation



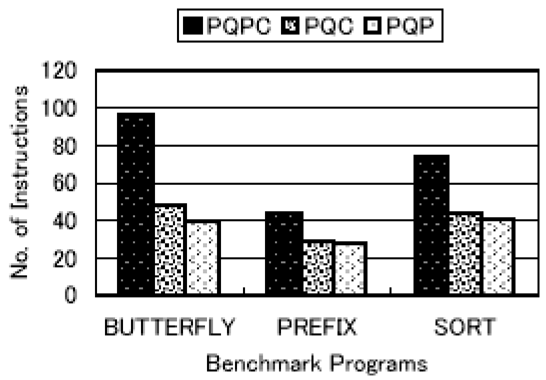
Breath first traversal

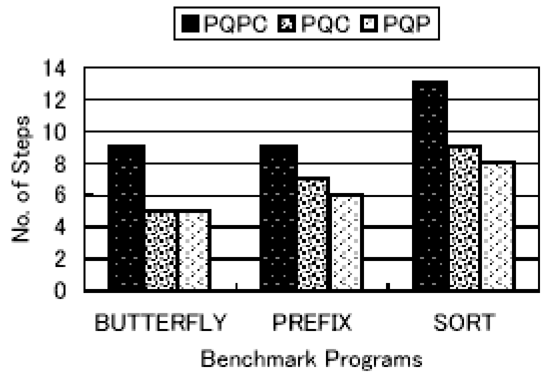


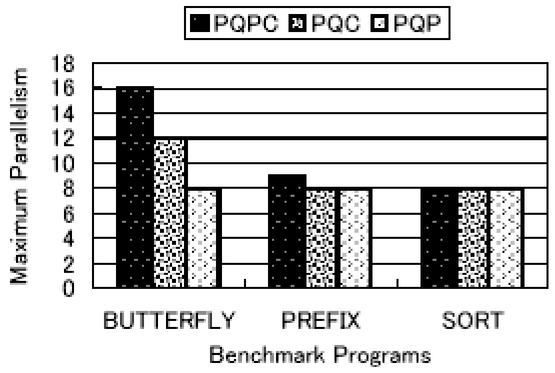
```
ld xo       add +2     st Y0
ld x1       add +2     st Y1
ld x2       sub -2     st Y2
ld x3       sub -2     st Y3
ld x4       add +2     st Y4
ld x5       add +2     st Y5
ld x6       sub -2     st Y6
ld x7       sub -2     st Y7
add +1      add +4
sub -1      add +4
add +1      add +4
sub -1      add +4
add +1      sub  -4
sub -1      sub -4
add +1      sub -4
sub -1      sub -4
```

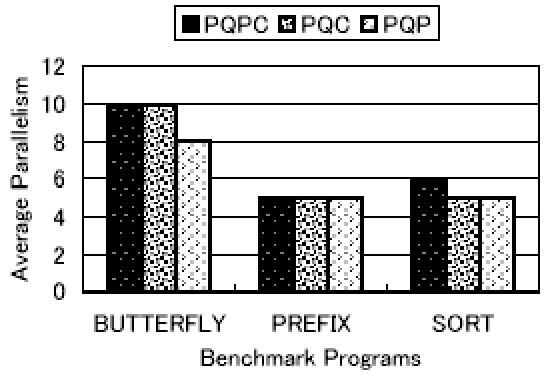
Generated assembly inst.

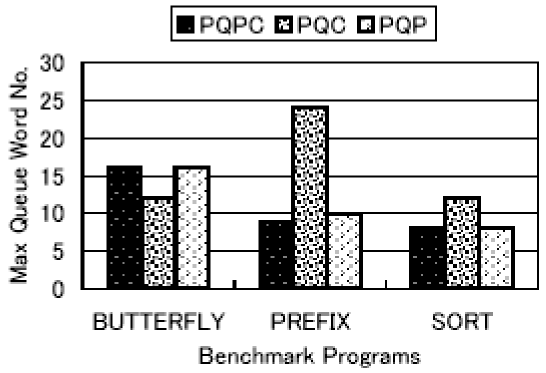
Queue Model
Hardware Parameters Estimation

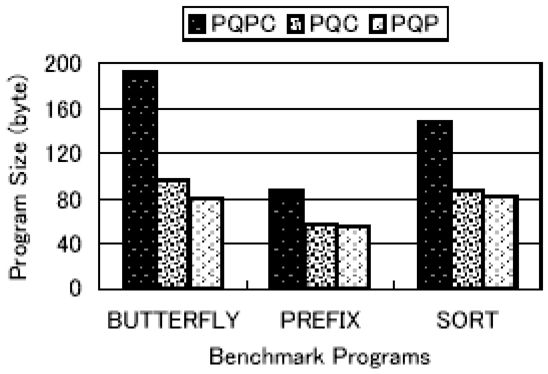










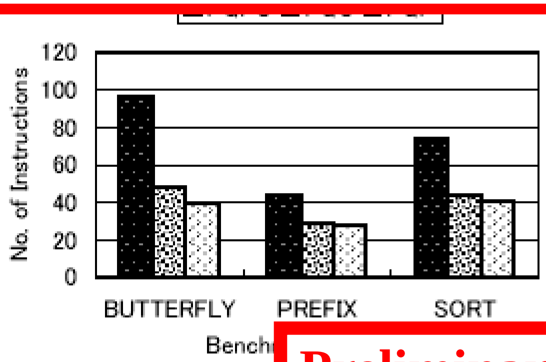


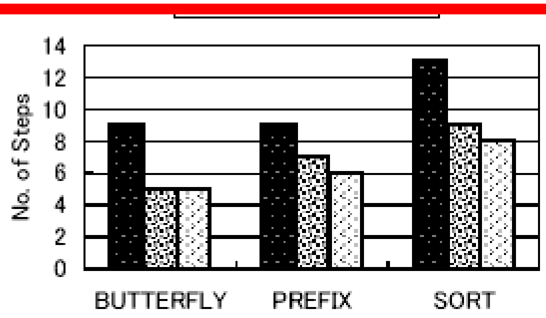
In average, produced order based processor performs better.

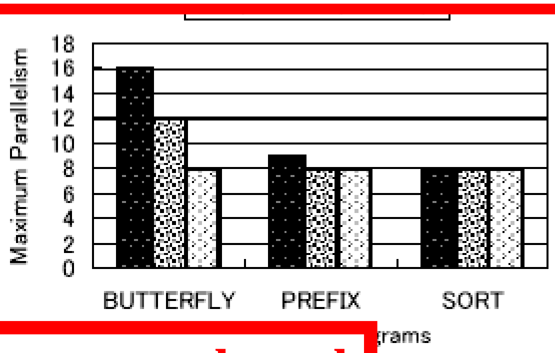
B. A. Abderazek, M. Arsenji, S. Shigeta, T. Yoshinaga, M. Sowa, **Queue Processor for Novel Queue Computing Paradigm Based on Produced Order Scheme**, IEEE computer Society, Int. Conf. On of High Performance Computing . pp. 169-177, July 2004.

Queue Model
Hardware Parameters Estimation

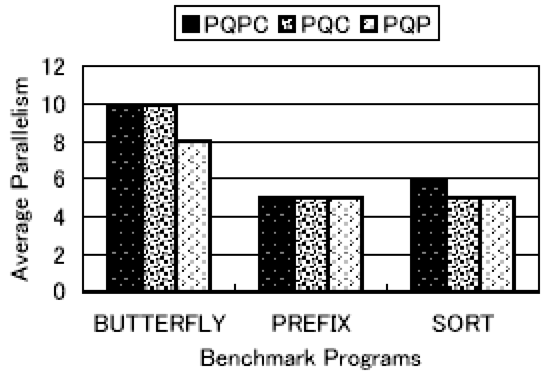
In average, PO based computing performs better.

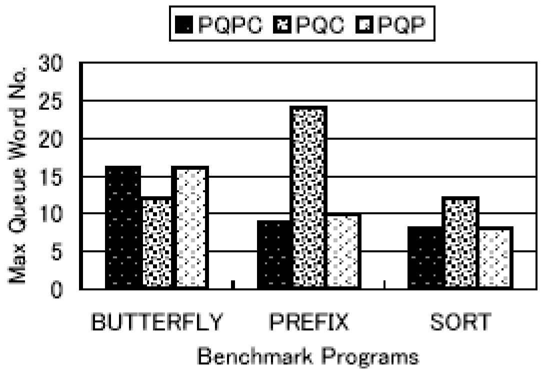


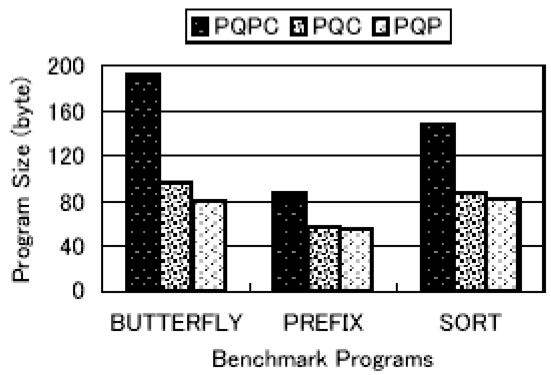




Preliminary hardware design parameters were selected





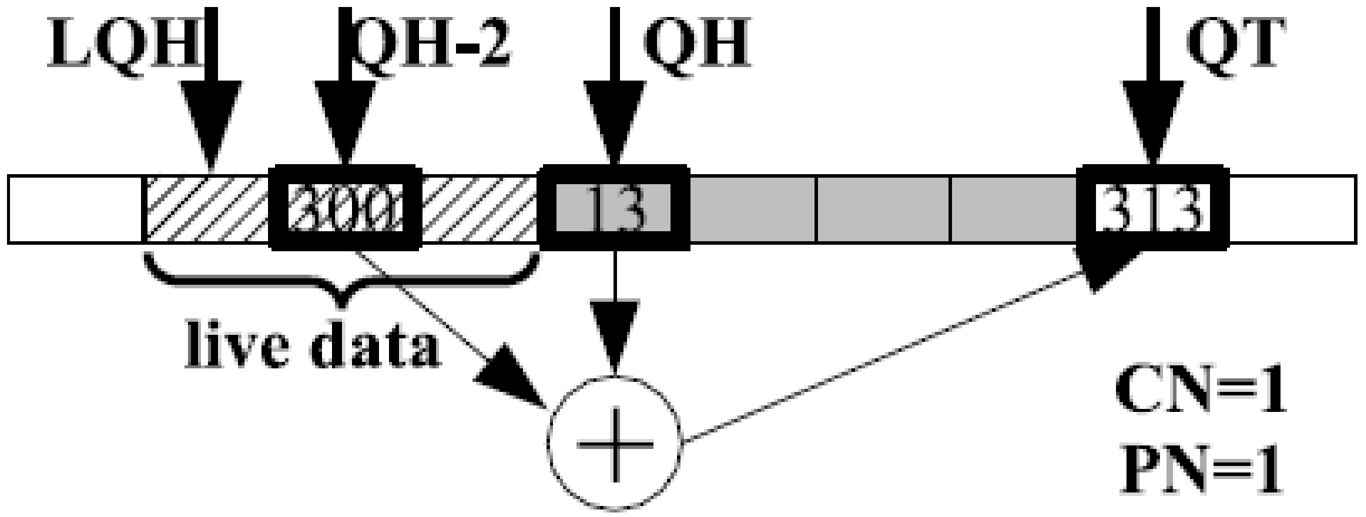


In average, produced order based processor performs better.

B. A. Abderazek, M. Arsenji, S. Shigeta, T. Yoshinaga, M. Sowa, **Queue Processor for Novel Queue Computing Paradigm Based on Produced Order Scheme**, IEEE computer Society, Int. Conf. On of High Performance Computing . pp. 169-177, uly 2004.

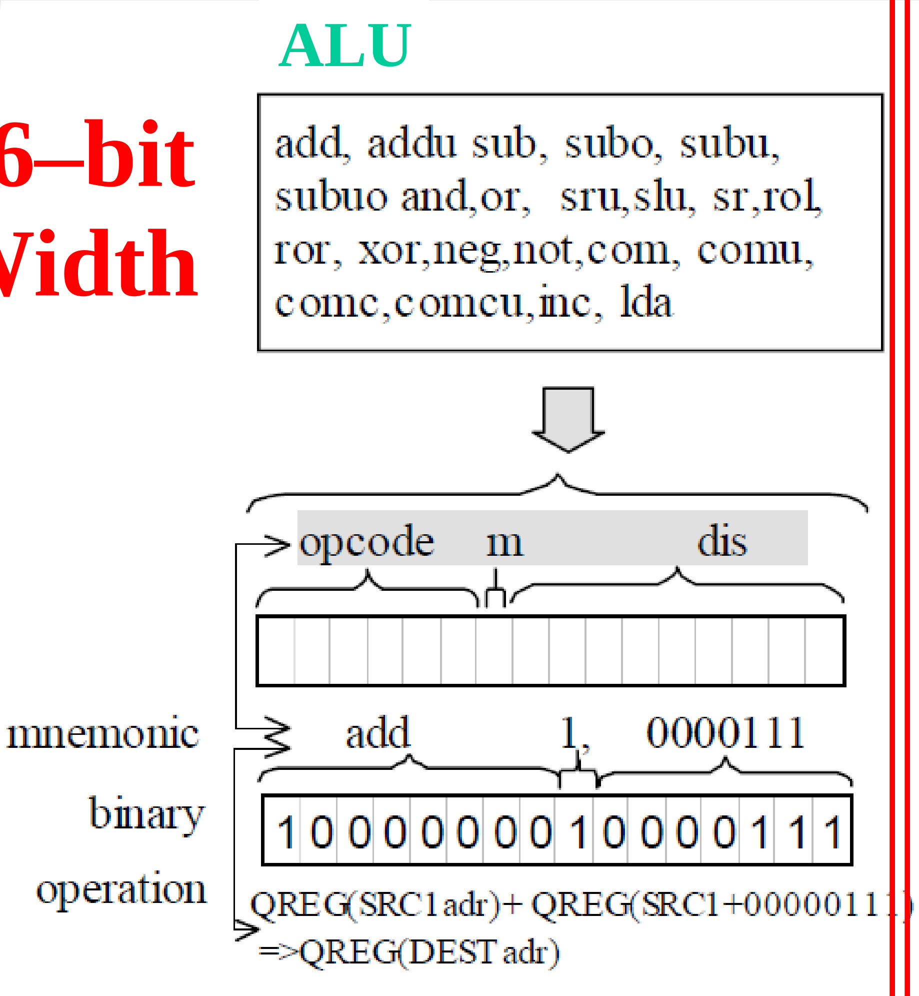
Produced Order Queue Model

b) produced order QCM: 'add -2'

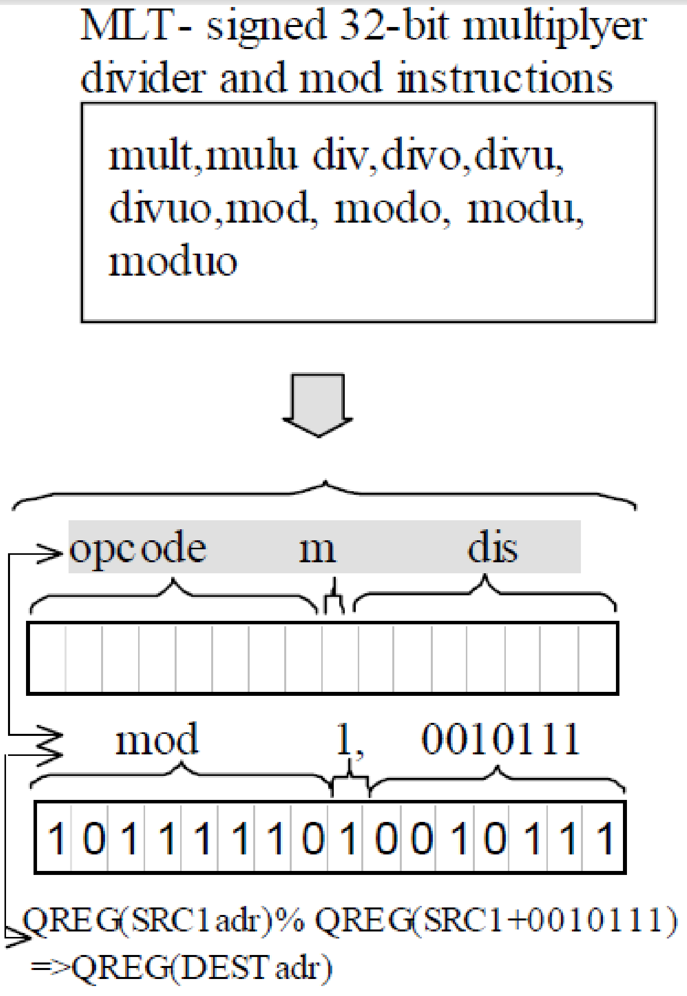


QueueCore architecture
Instruction set architecture

16–bit Width

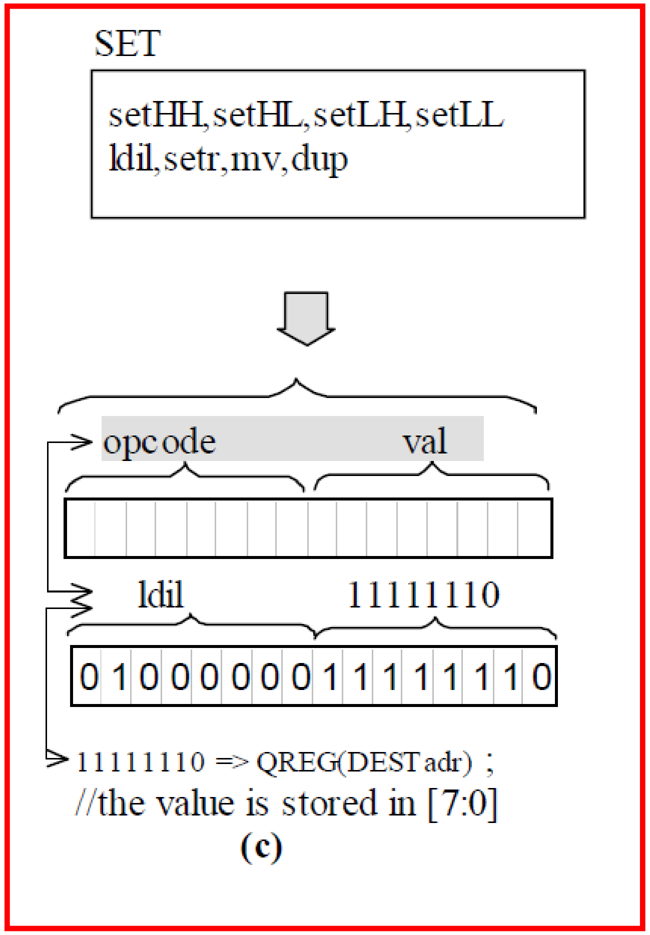

ALU
add, addu sub, subo, subu, subuo and,or, sru,slu, sr,rol, ror, xor,neg,not,com, comu, comc,comcu,inc, lda
opcode
m
dis
mnemonic
add 1, 0000111
binary
1 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1
operation
QREG(SRC1adr)+ QREG(SRC1+0000111) =>QREG(DEST adr)

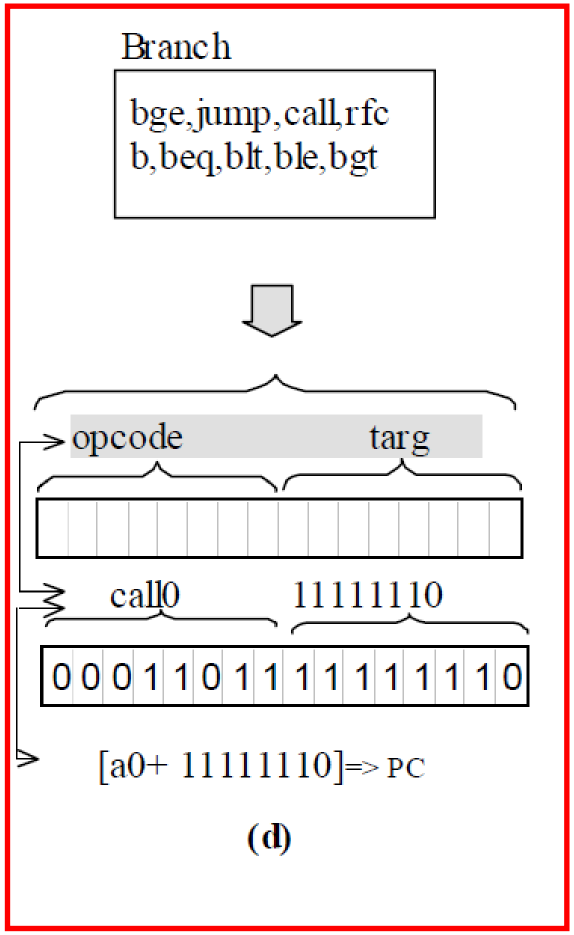
(a)

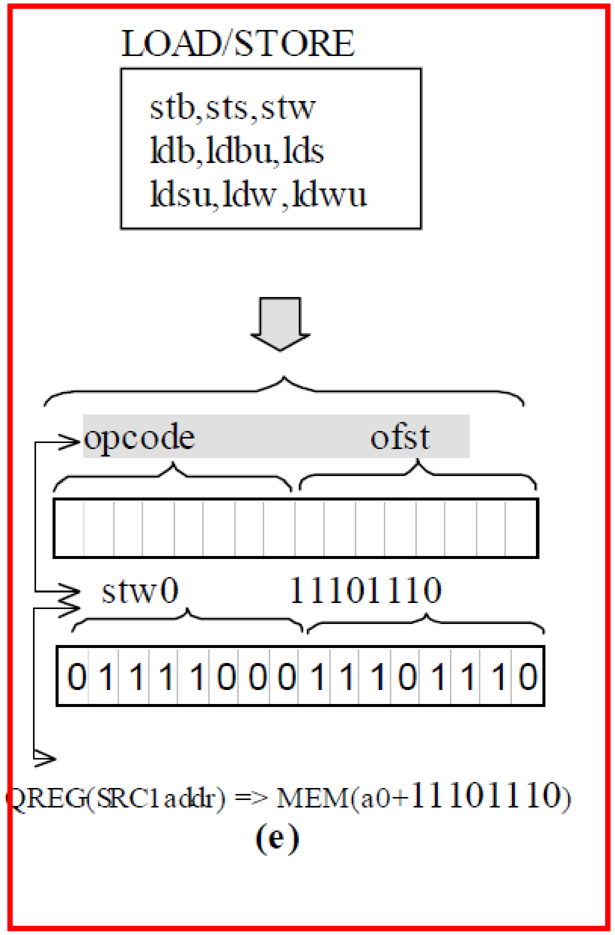

MLT- signed 32-bit multiplyer divider and mod instructions
mult,mulu div,divo,divu, divuo,mod, modo, modu, moduo
opcode
m
dis
mod 1, 0010111
1 0 1 1 1 1 1 0 1 0 0 1 0 1 1 1
QREG(SRC1adr)% QREG(SRC1+0010111) =>QREG(DEST adr)

(b)

QueueCore architecture
Instruction set architecture







• *B. A. Abderazek, S. Shigeta, Y. Tsutomu and S. Masahiro,* ***Reduced Bit-Width Instruction Set Architecture for Q-mode, Execution****, IPSJ Arch. Conf. pp. 19-23, June 2003*

QueueCore architecture

Instruction set architecture: Offset/Disp. extension

```
int main (void)
 {
    int a[1000];
    int i,x,y;

    if (y = =1) {
                x = a[i];
     }
     else {
             x = (x*2) +1000;
     }
}
```

(a)

```
main:  ld  4000(d)
       ldil  1       ;load immediate 1 to QT
       ceq           ;compare QH and QH+1 value
       bt       L1    and check equality
L0:    ld 4008 (d)
       ldi  4        ;load immediate 1 to QT
       mvrq          ;move value from register to QT
       ldil  0       ;load immediate 0 to QT
       add           ;add QH and QH+1 value and send
       mul            the result to QT
       add
       st 4004 (d)
       Jump L2
L1:    ld 4004 (d)
       ldil  2
       mul           ;multiply QH and QH+1 value
       ldil 1000      and send  the result to QT
       add
       st 4004(d)
L2: mvrq
       ld  4028 (d)
       ld  4024 (d)
       add
       jump 10(a)
```

(b)

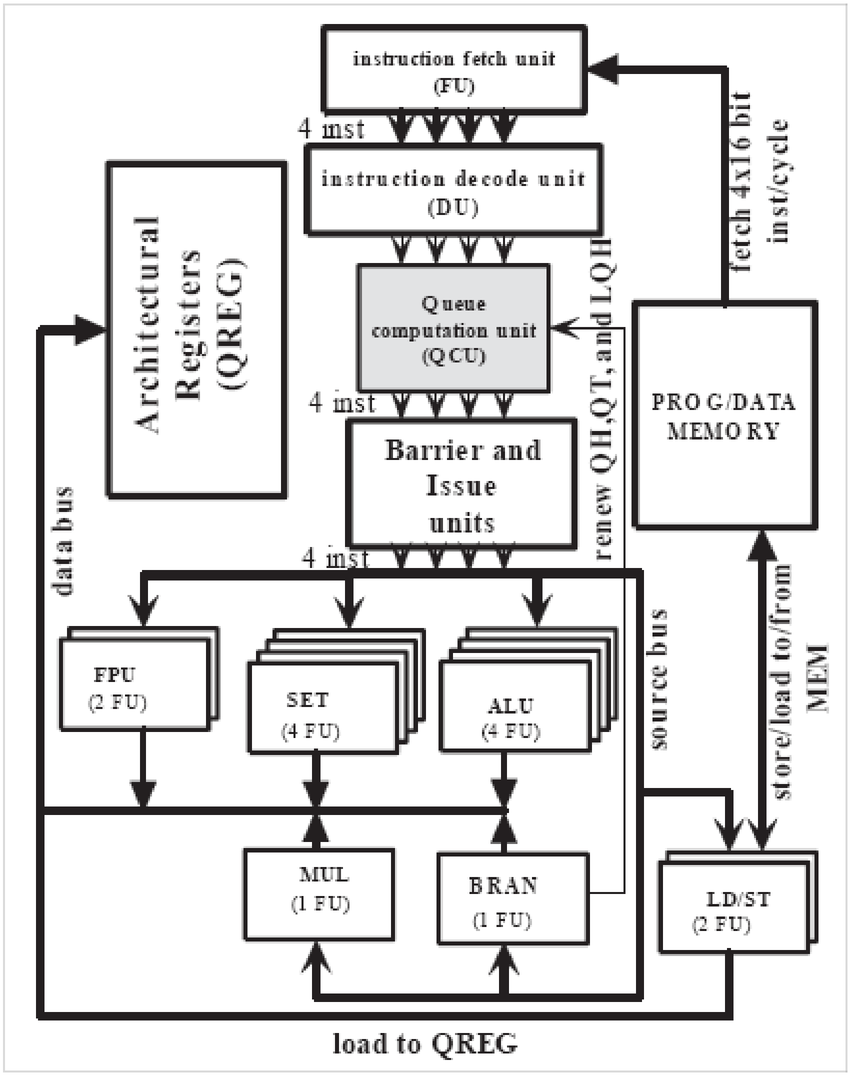
Without "convop"

```
main: covop 62
      ld  32(d)
      ldil  1
      ceq
      bt        L1
L0:   covop 15
      ld 62 (d)
      ldi  40
      mvrq
      ldil  0
      add
      mul
      add
      covop 62
      st 36 (d)
      Jump L2
L1:   covop 62
      ld 36 (d)
      ldil  2
      mul
      covop 15
      ldil 40
      add
      covop 62
      st 36(d)
L2: mvrq
      covop 62
      ld  60(d)
      covop 62
      ld  56(d)
      add
      jump   10(a)
```

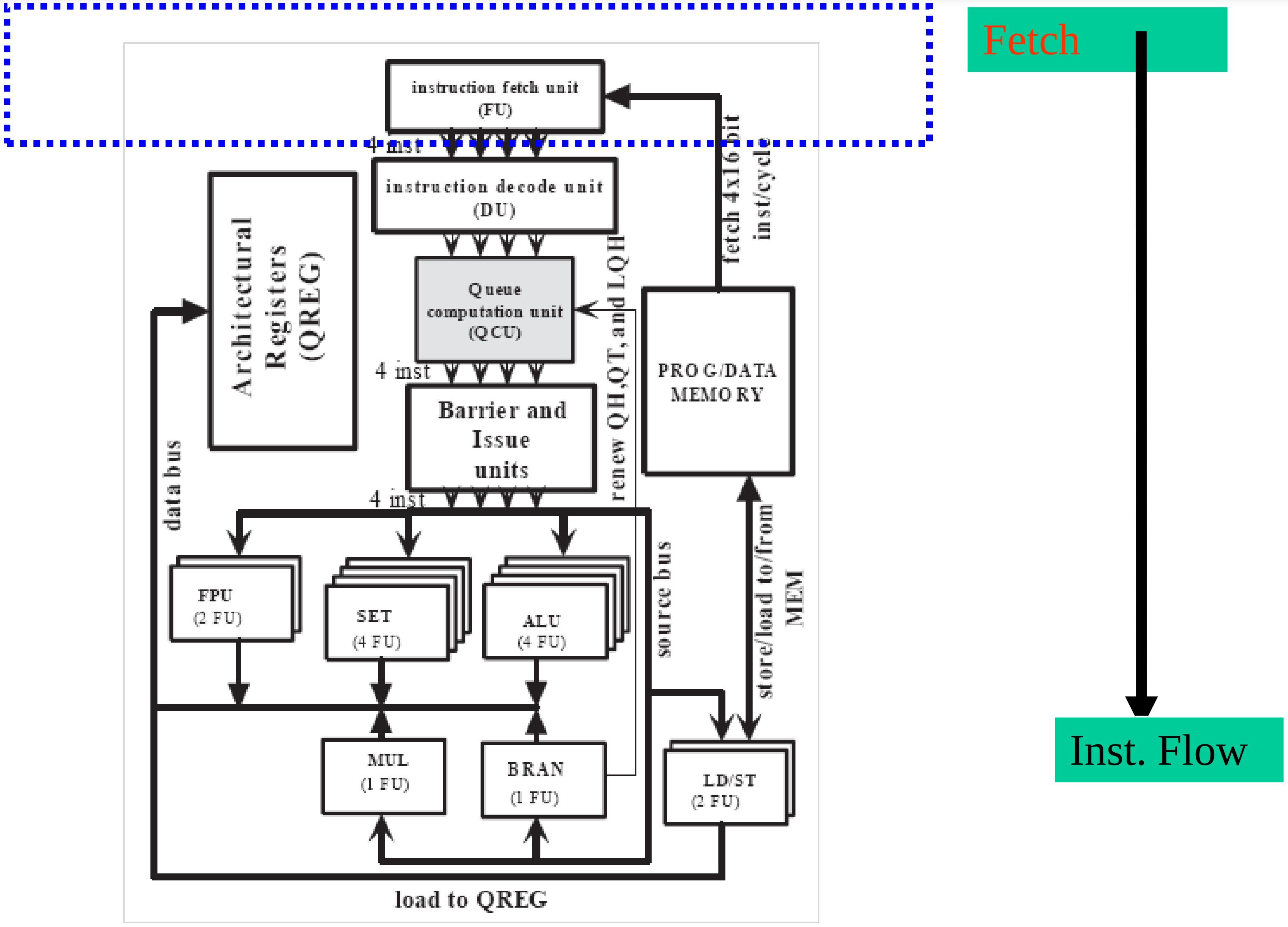
(c)

With "convop"

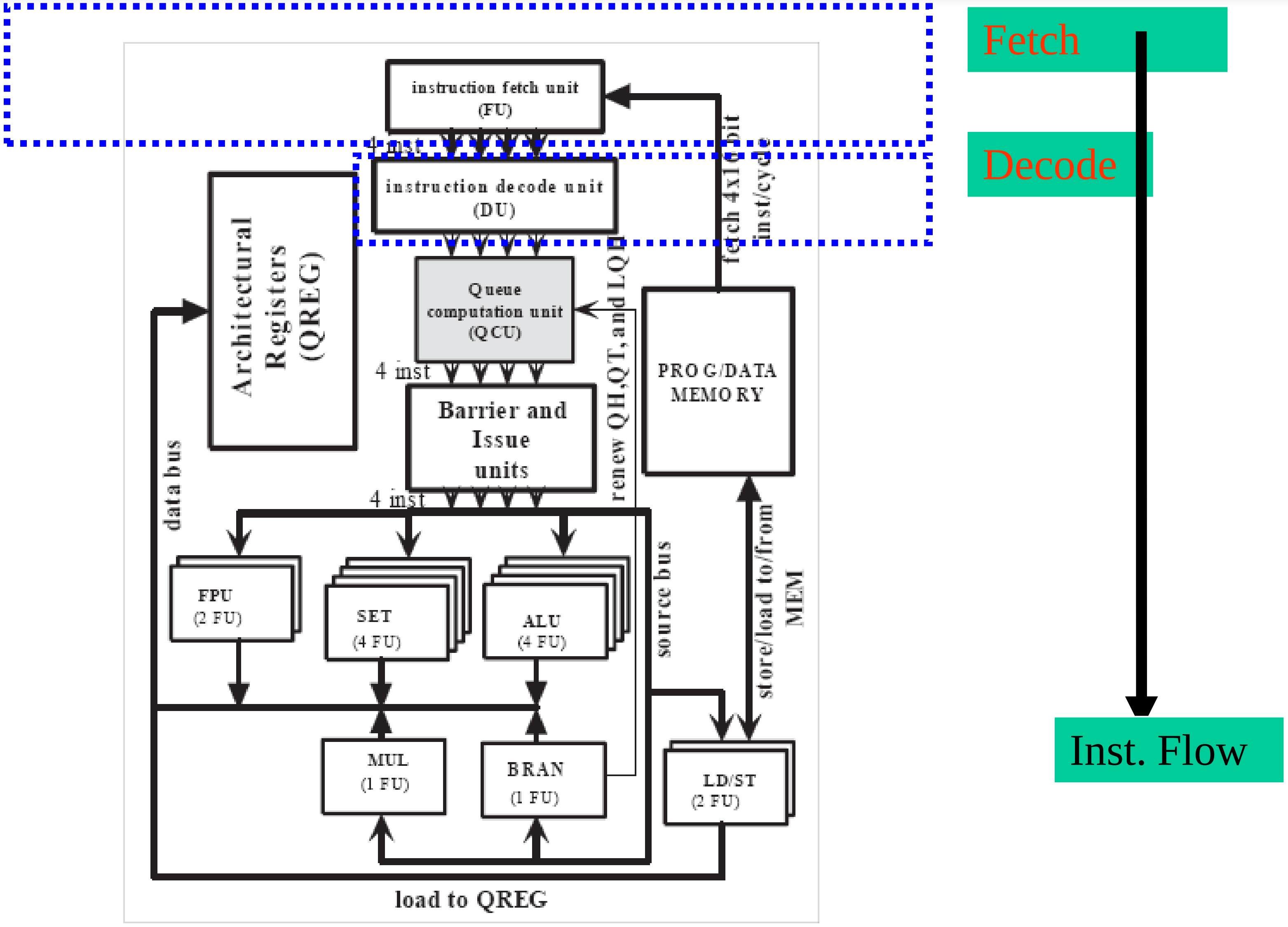
QueueCore architecture
Data path



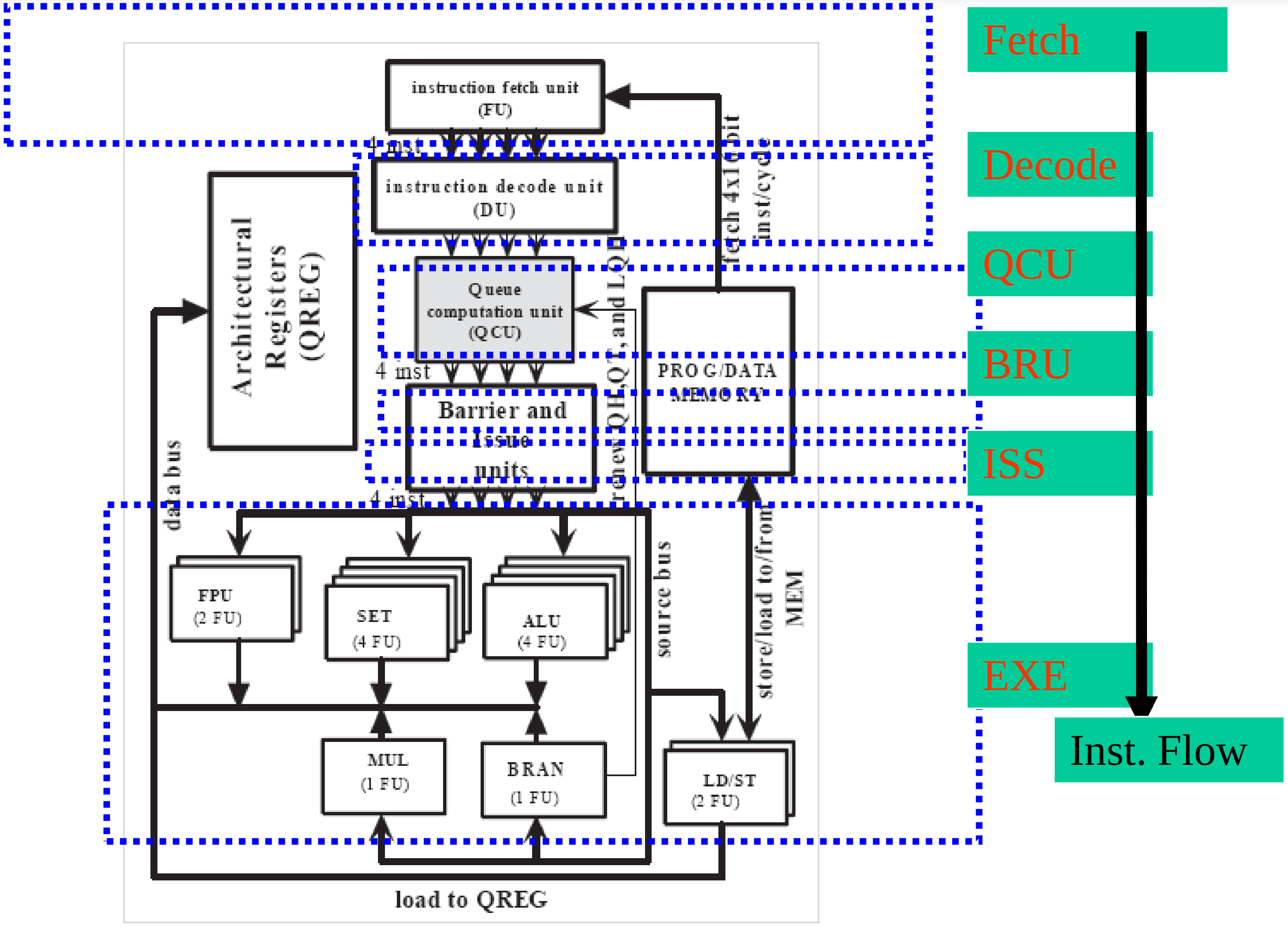
QueueCore architecture
Data path



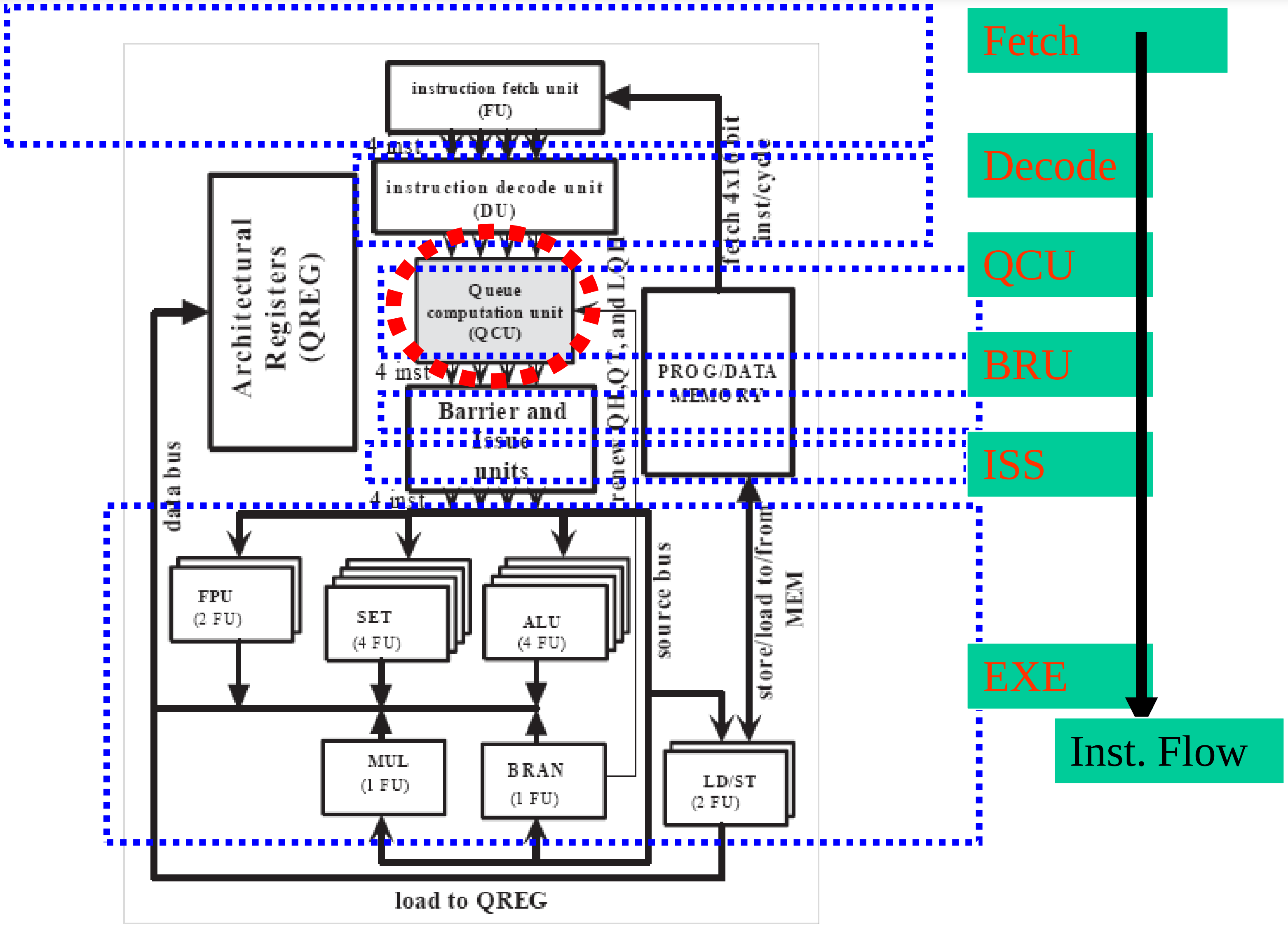
QueueCore architecture
Data path



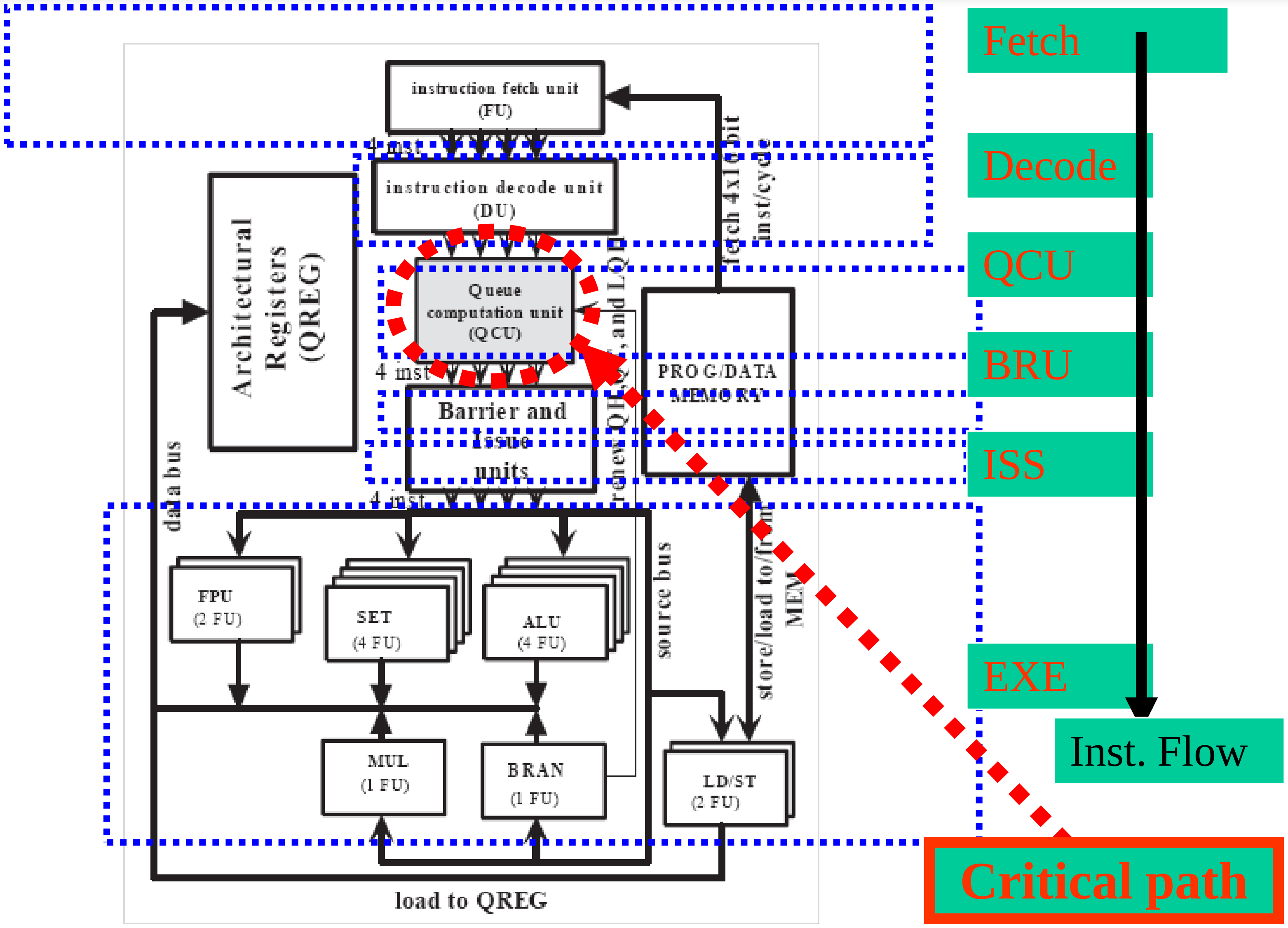
QueueCore architecture
Data path



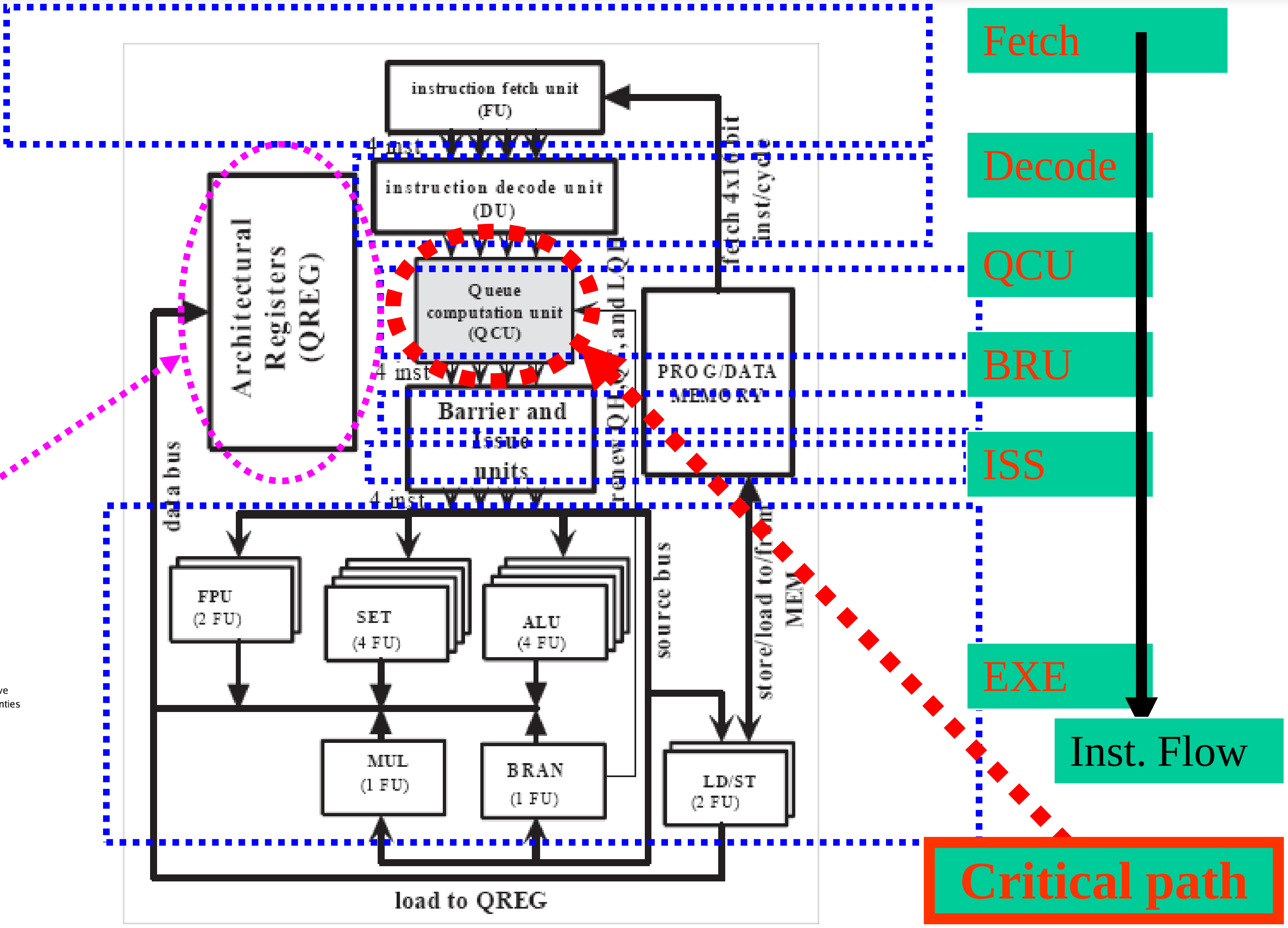
QueueCore architecture
Data path



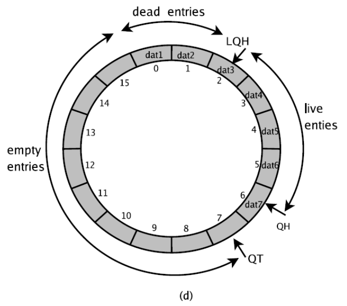
QueueCore architecture
Data path



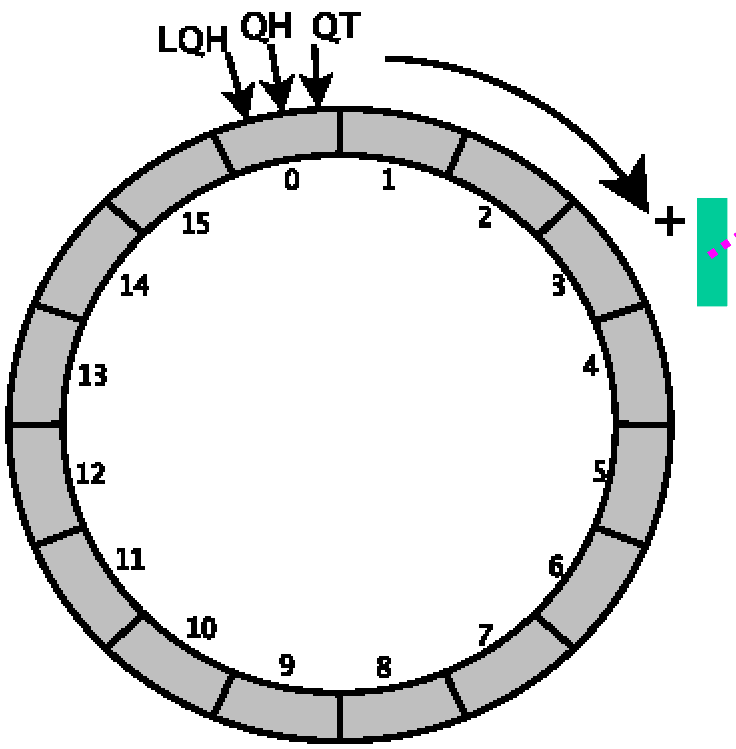
QueueCore architecture
Data path

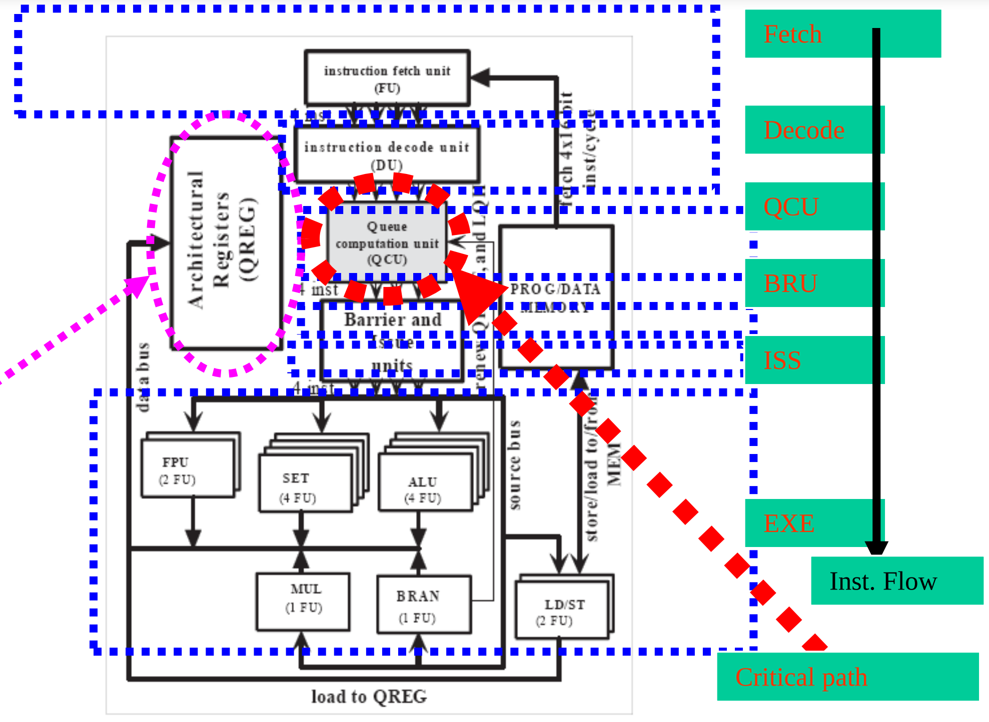


Circular Queue Register



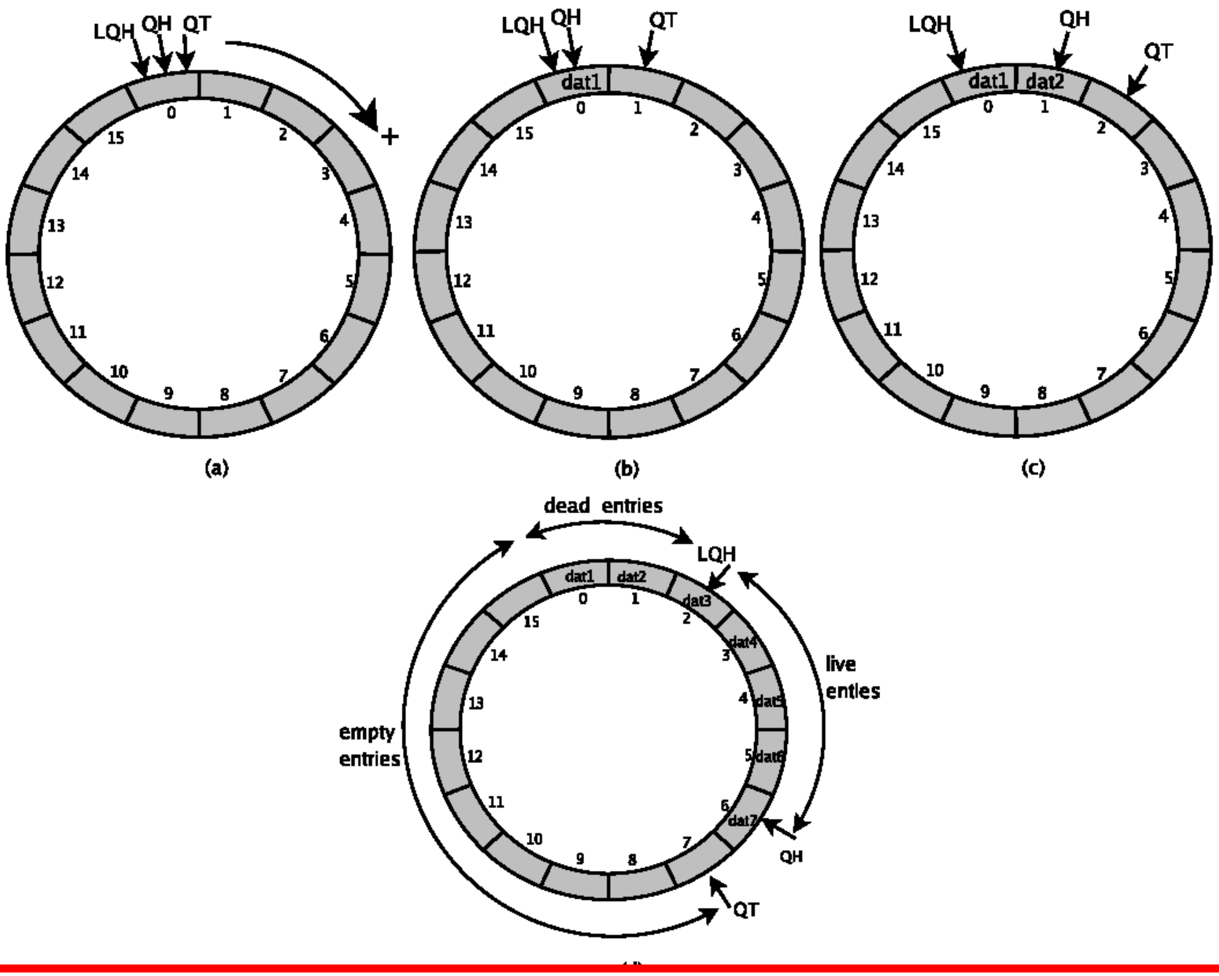
QueueCore architecture
Data path





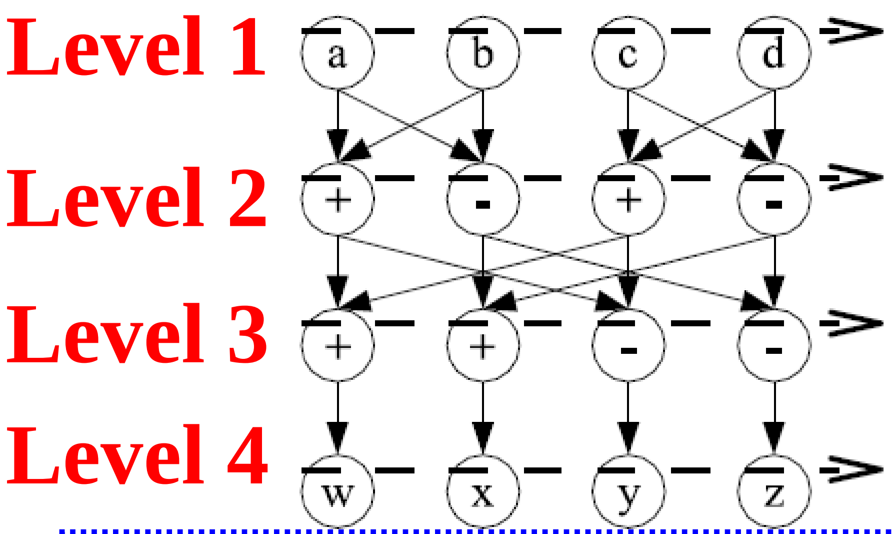
Circular Queue Register (QREG)

QueueCore architecture
Circular Queue Register Entries



QH, QT, LQH and QREG entries

QueueCore architecture
Grouped ILP

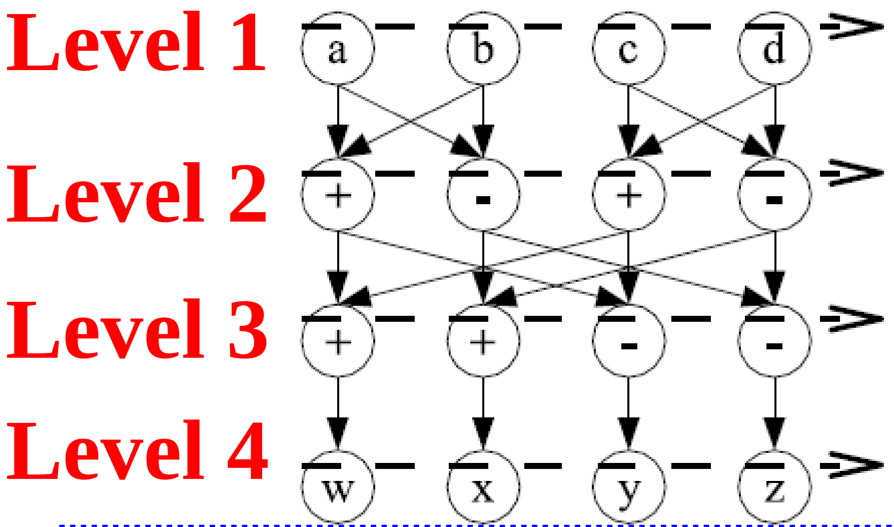


Grouped ILP

4 inst. in each level

```
Group 1:    ld a; ld b; ld c; ld d;

Group 2:    add +1; sub -1; add +1; sub -1;

Group 3:    add +2; add +2; sub -2; sub -2;

Group 4:    st w; st x; st y; st z;
```

QueueCore architecture
Grouped ILP



4 inst. in each level

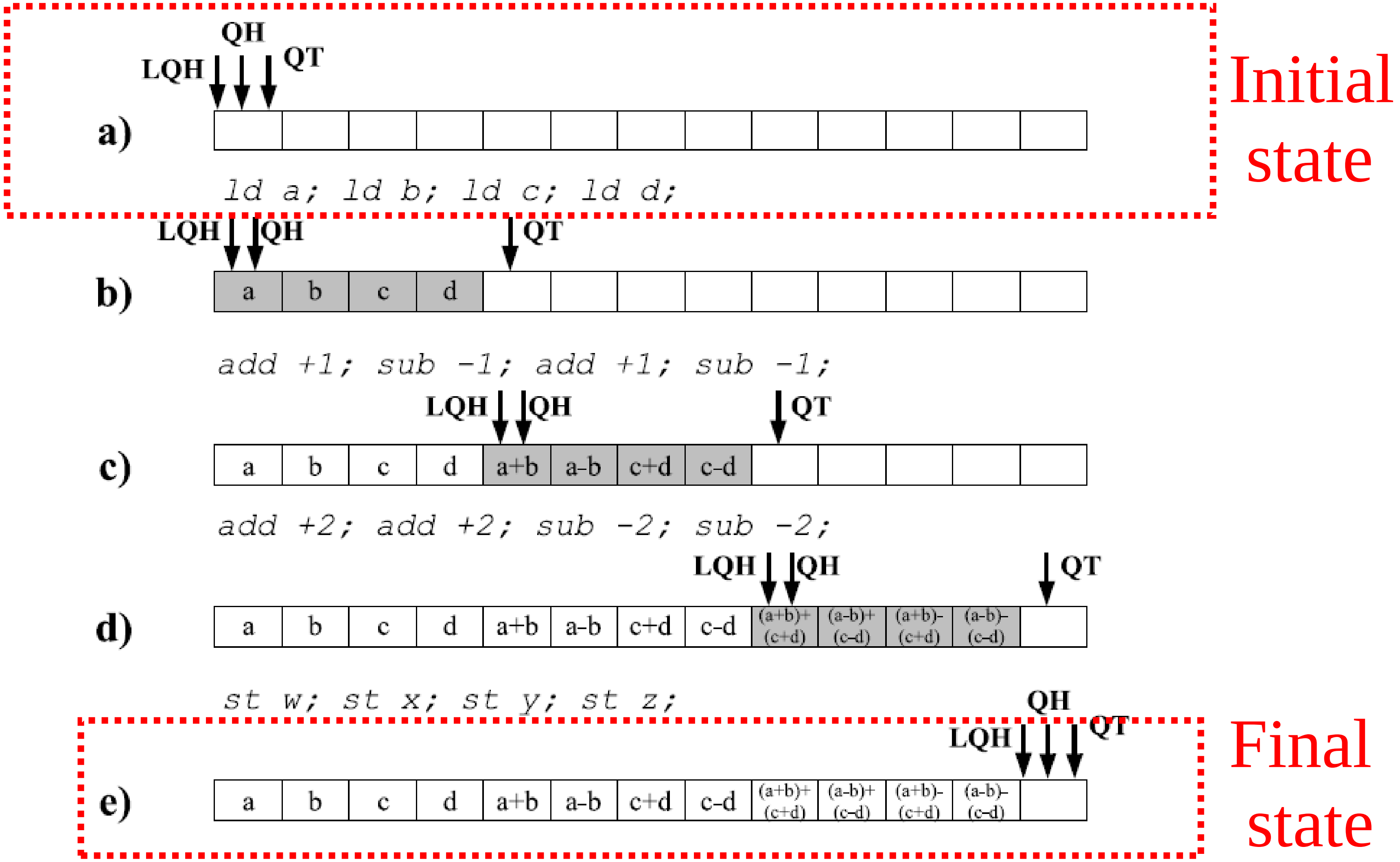
```
Group 1:    ld a; ld b; ld c; ld d;

Group 2:    add +1; sub -1; add +1; sub -1;

Group 3:    add +2; add +2; sub -2; sub -2;

Group 4:    st w; st x; st y; st z;
```

Grouped ILP

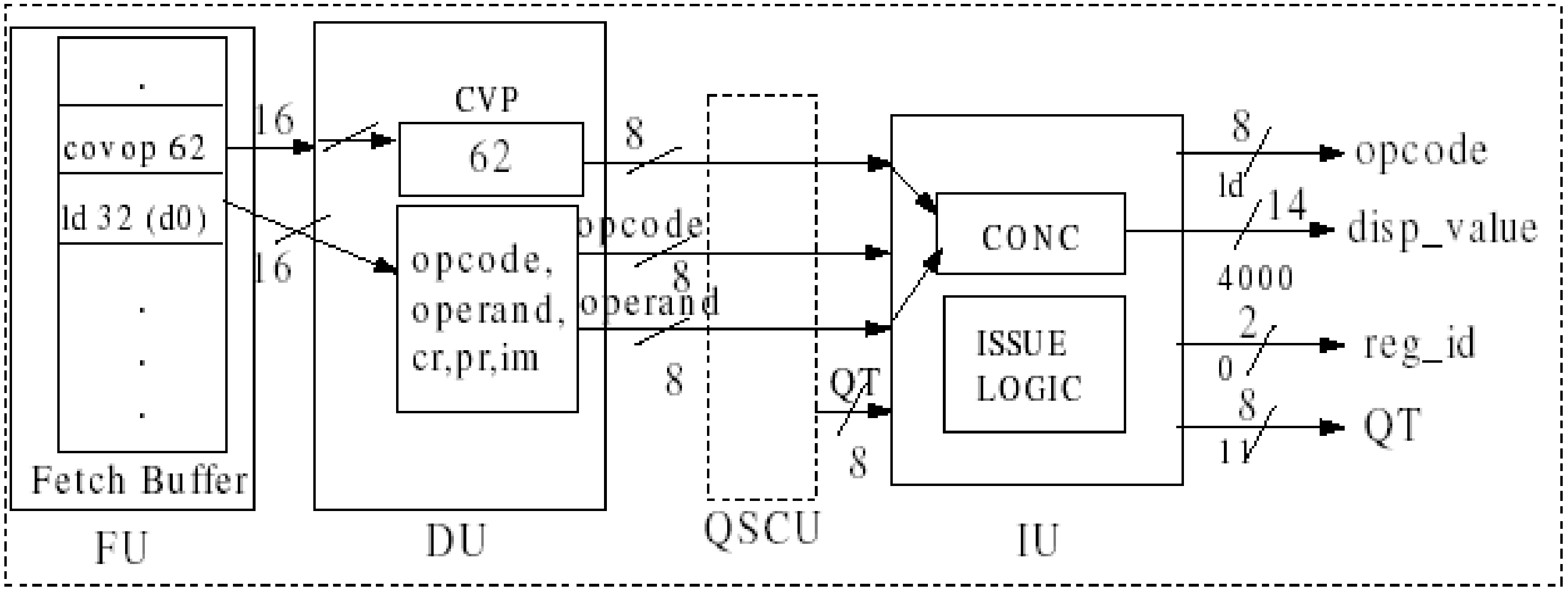
small IWB

QueueCore architecture

QREG pointers manipulations



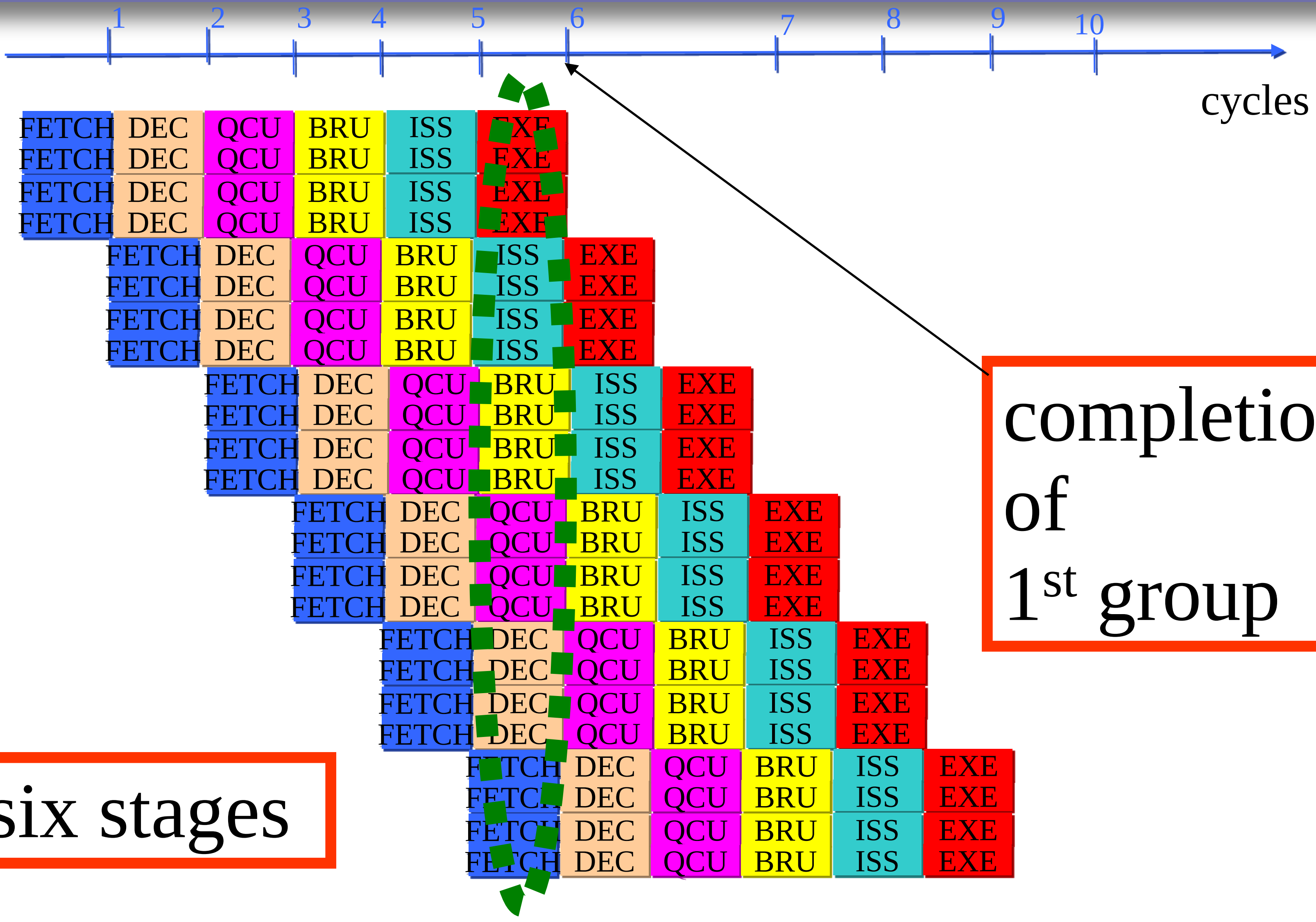
QueueCore architecture
Offset extension scheme



Concatenate the value in the CVP with the displacement value

QueueCore architecture

Pipeline stages

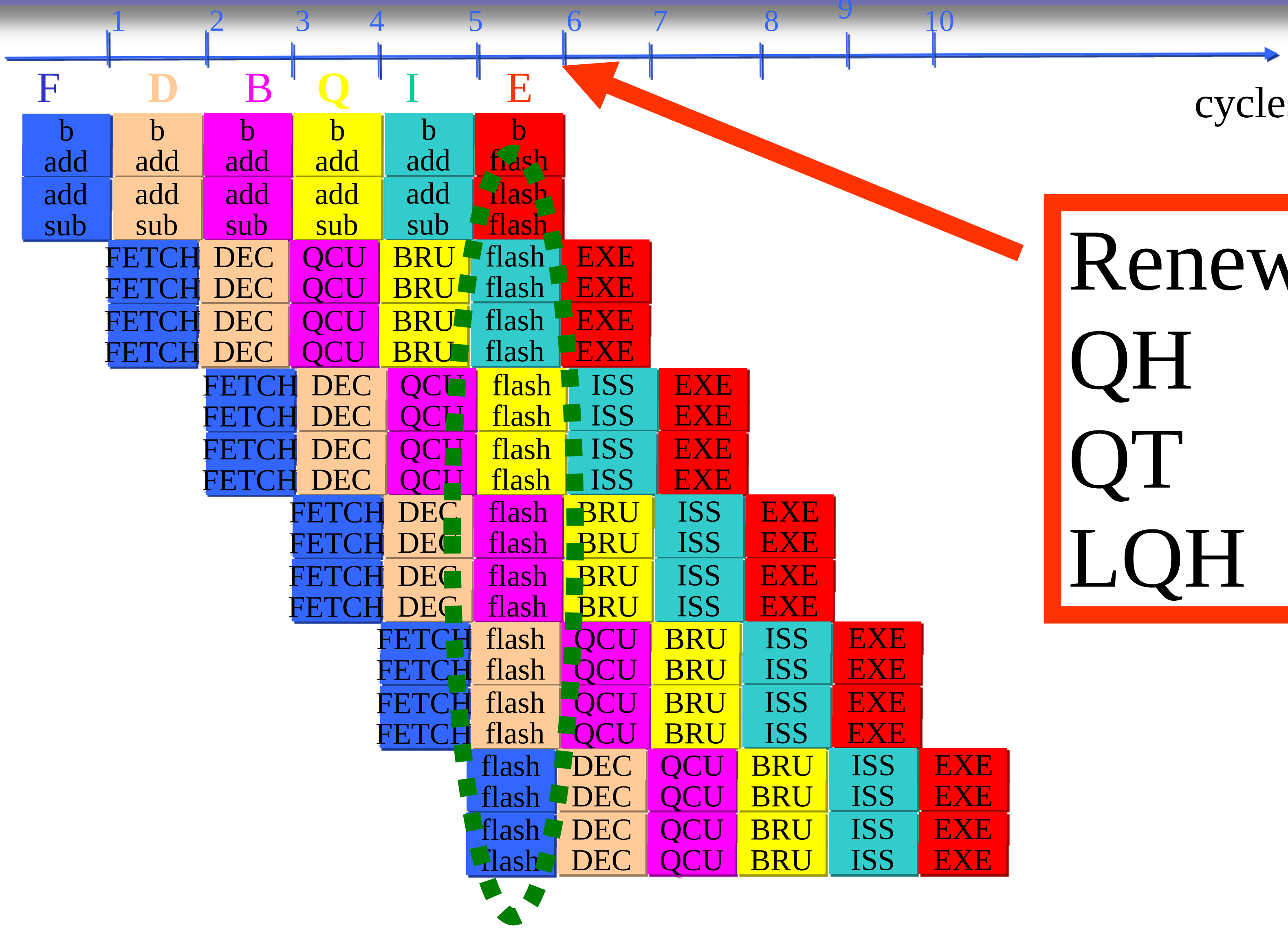


completion of 1st group

six stages

QueueCore architecture

Branch taken handling

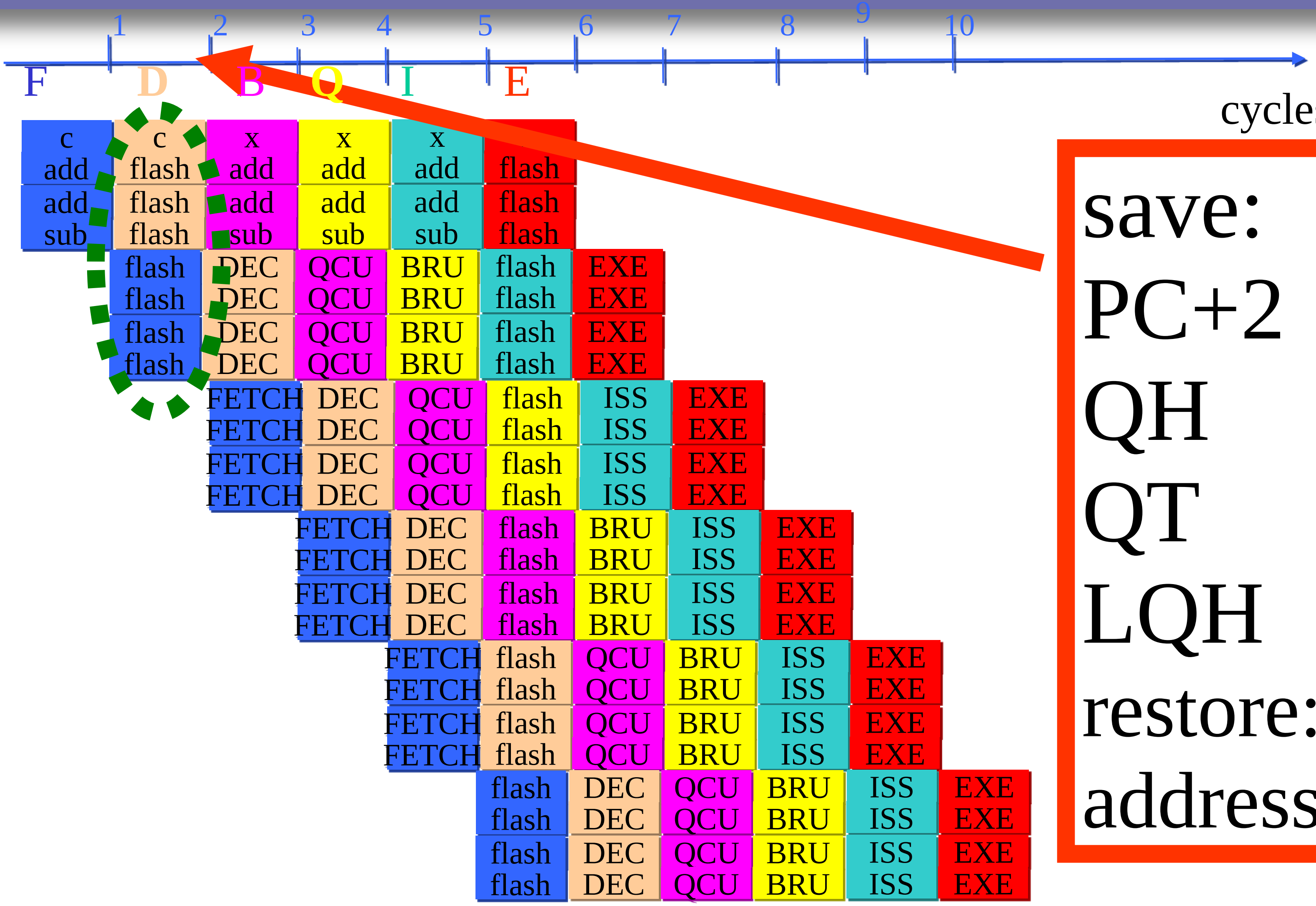


1 2 3 4 5 6 7 8 9 10
cycles
F D B Q I E
b b b b b b
add add add add add flash
add add add add add flash
sub sub sub sub sub flash
FETCH DEC QCU BRU flash EXE
FETCH DEC QCU BRU flash EXE
FETCH DEC QCU BRU flash EXE
FETCH DEC QCU BRU flash EXE
FETCH DEC QCU flash ISS EXE
FETCH DEC QCU flash ISS EXE
FETCH DEC QCU flash ISS EXE
FETCH DEC QCU flash ISS EXE
FETCH DEC flash BRU ISS EXE
FETCH DEC flash BRU ISS EXE
FETCH DEC flash BRU ISS EXE
FETCH DEC flash BRU ISS EXE
FETCH flash QCU BRU ISS EXE
FETCH flash QCU BRU ISS EXE
FETCH flash QCU BRU ISS EXE
FETCH flash QCU BRU ISS EXE
flash DEC QCU BRU ISS EXE
flash DEC QCU BRU ISS EXE
flash DEC QCU BRU ISS EXE
flash DEC QCU BRU ISS EXE

Renew:
QH
QT
LQH

QueueCore architecture

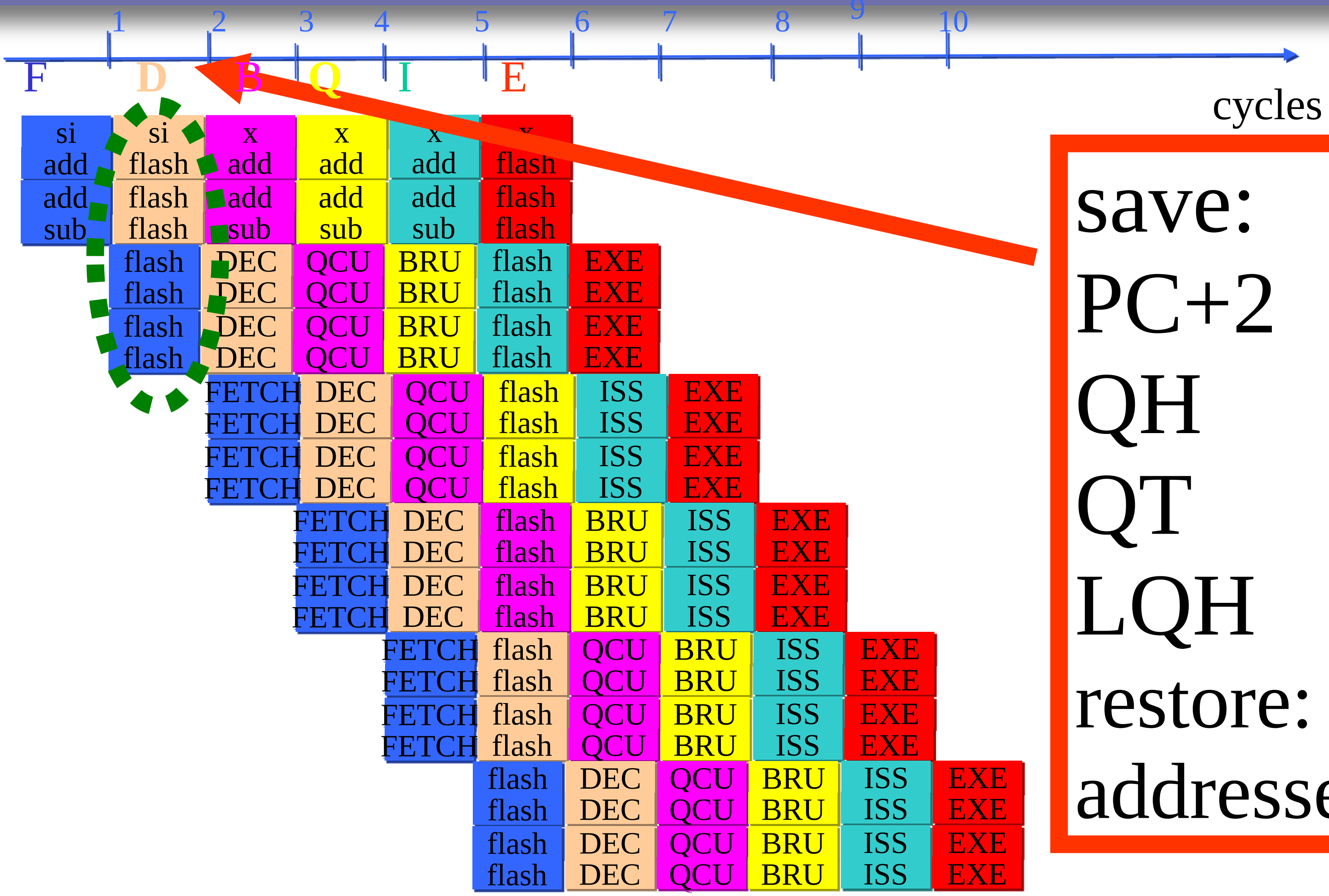
Call handling



save:
PC+2
QH
QT
LQH
restore:
addresses

QueueCore architecture

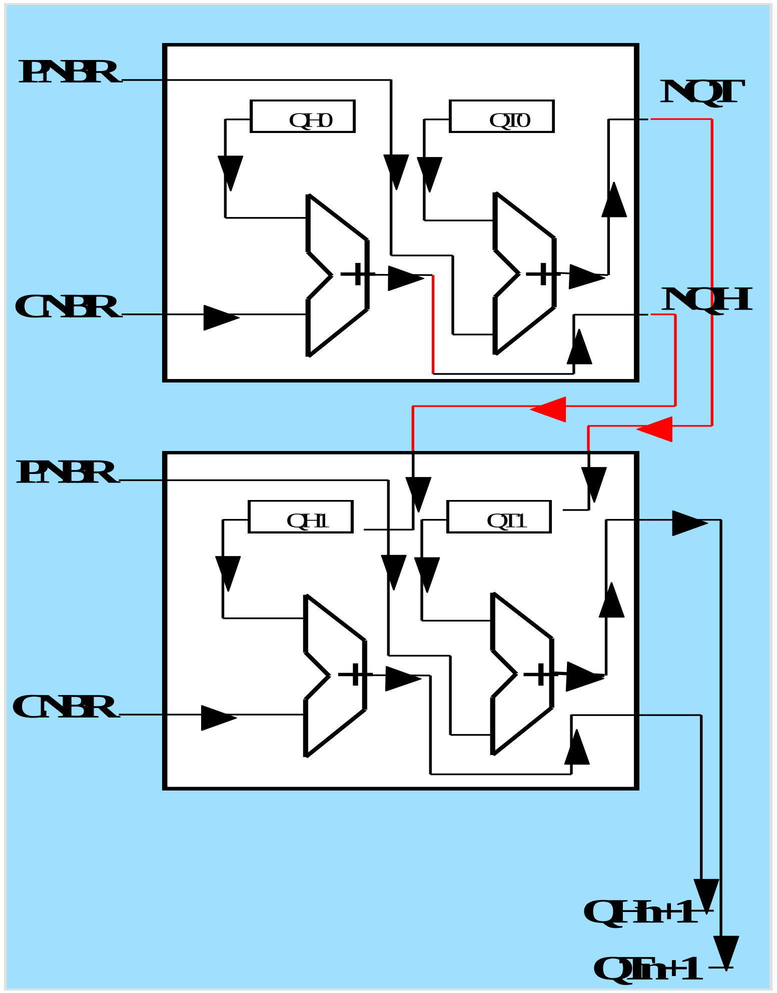
Interrupt handling


1 2 3 4 5 6 7 8 9 10
cycles
F D B Q I E
si add add sub | si flash flash flash | x add add sub | x add add sub | x add add sub | x flash flash flash
flash flash flash flash | DEC DEC DEC DEC | QCU QCU QCU QCU | BRU BRU BRU BRU | flash flash flash flash | EXE EXE EXE EXE
FETCH FETCH FETCH FETCH | DEC DEC DEC DEC | QCU QCU QCU QCU | flash flash flash flash | ISS ISS ISS ISS | EXE EXE EXE EXE
FETCH FETCH FETCH FETCH | DEC DEC DEC DEC | flash flash flash flash | BRU BRU BRU BRU | ISS ISS ISS ISS | EXE EXE EXE EXE
FETCH FETCH FETCH FETCH | flash flash flash flash | QCU QCU QCU QCU | BRU BRU BRU BRU | ISS ISS ISS ISS | EXE EXE EXE EXE
flash flash flash flash | DEC DEC DEC DEC | QCU QCU QCU QCU | BRU BRU BRU BRU | ISS ISS ISS ISS | EXE EXE EXE EXE

save:
PC+2
QH
QT
LQH
restore:
addresses

QueueCore architecture
Queue Computation

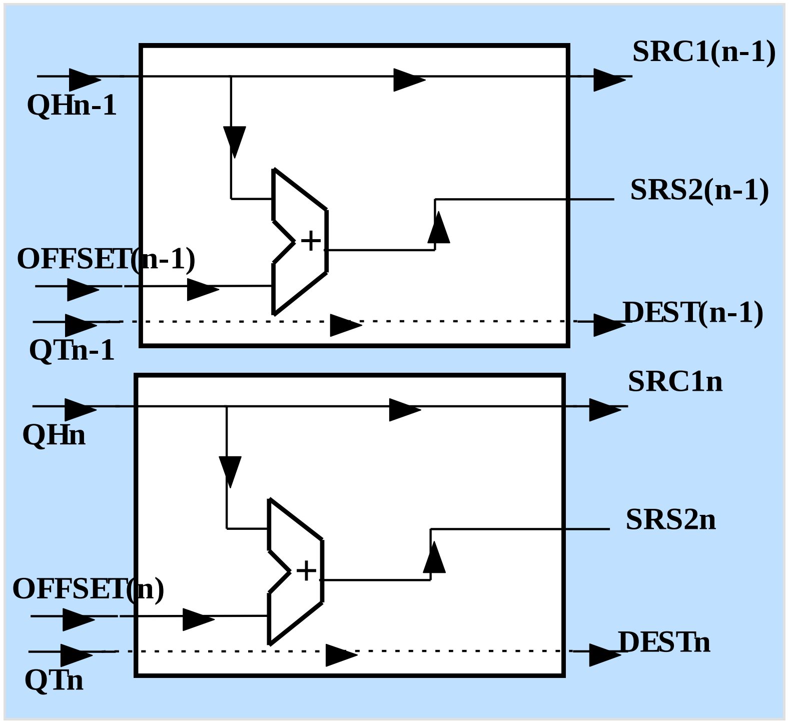
Next QH and QT values calculation



$$LQH_{i+1} = LQH_i + CN_{inst}$$
$$QH_{i+1} = QH_i + CN_{inst}$$
$$QT_{i+1} = QT_i + PN_{inst}$$

QueueCore architecture
Queue Computation

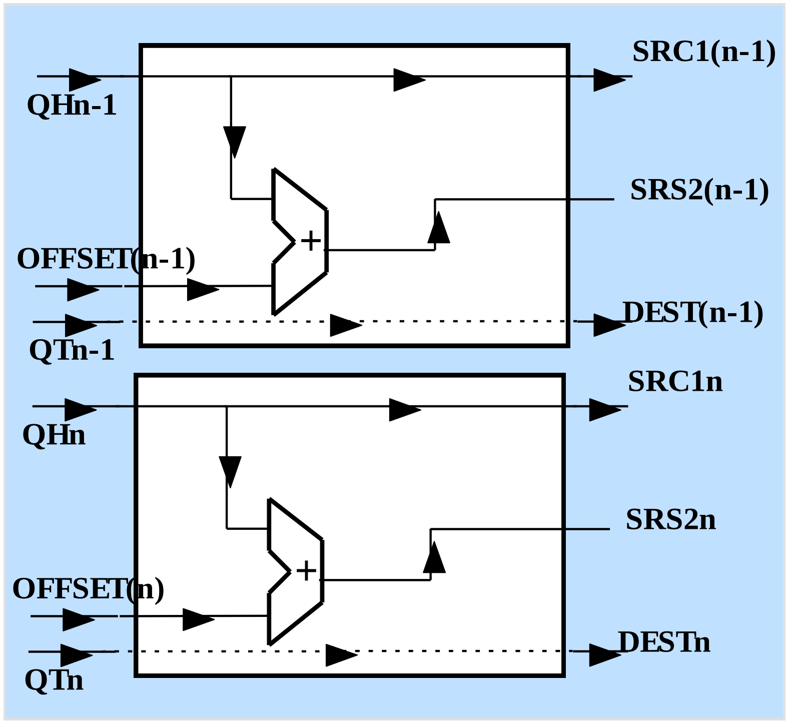
Source 1 and Source 2 address calculations



OFFSET: positive/negative integer value that indiactes the location of SRC2(n-1) from the QH(n-1)
QTn : queue tail value of instruction n
DESTn : destination location of instruction n

QueueCore architecture
Queue Computation

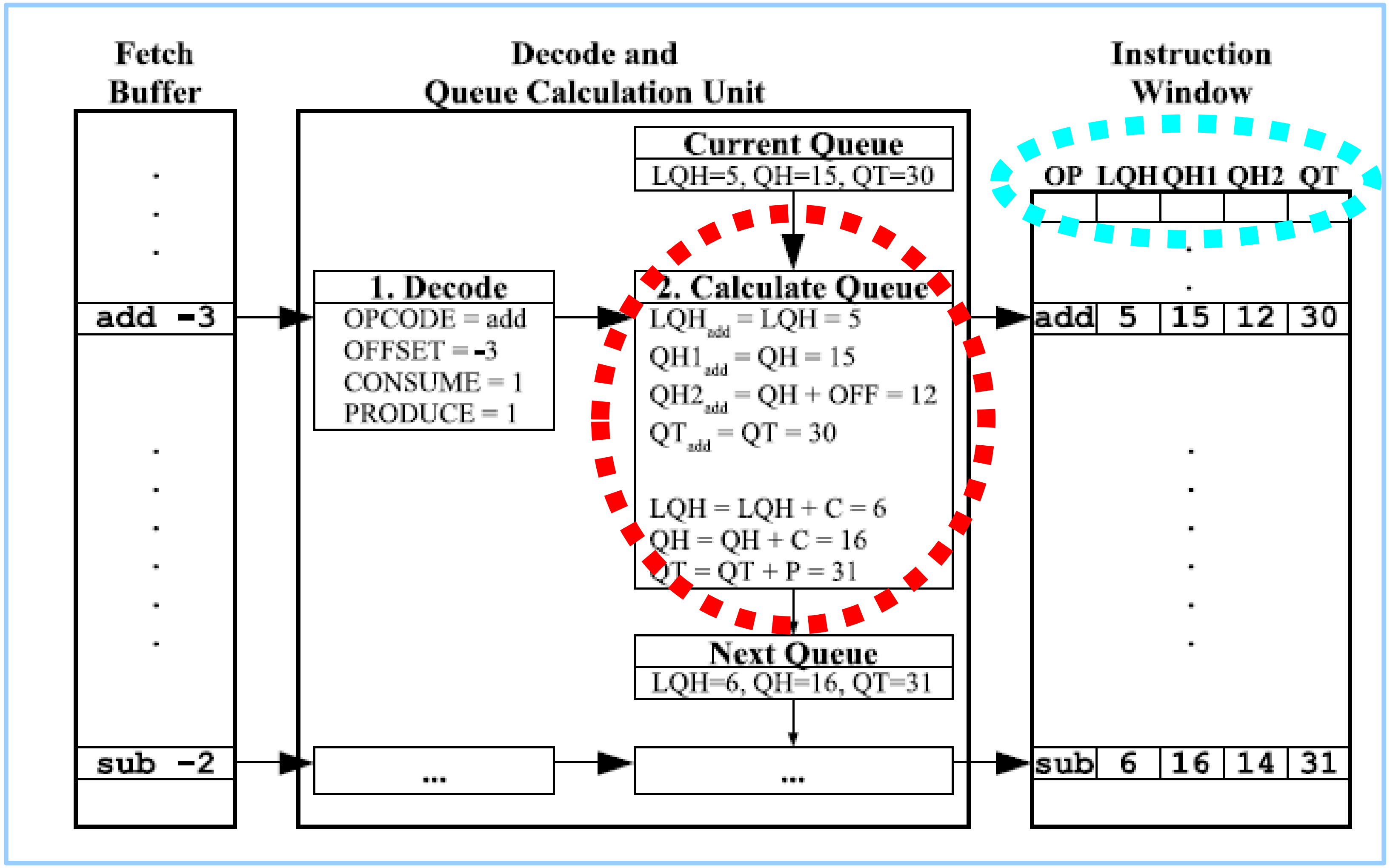
Source 1 and Source 2 address calculations



$$
\begin{aligned}
LQH_{inst} &= LQH_i \\
QH1_{inst} &= QH_i \\
QH2_{inst} &= QH_i + OFFSET_{inst} \\
QT_{inst} &= QT_i
\end{aligned}
$$

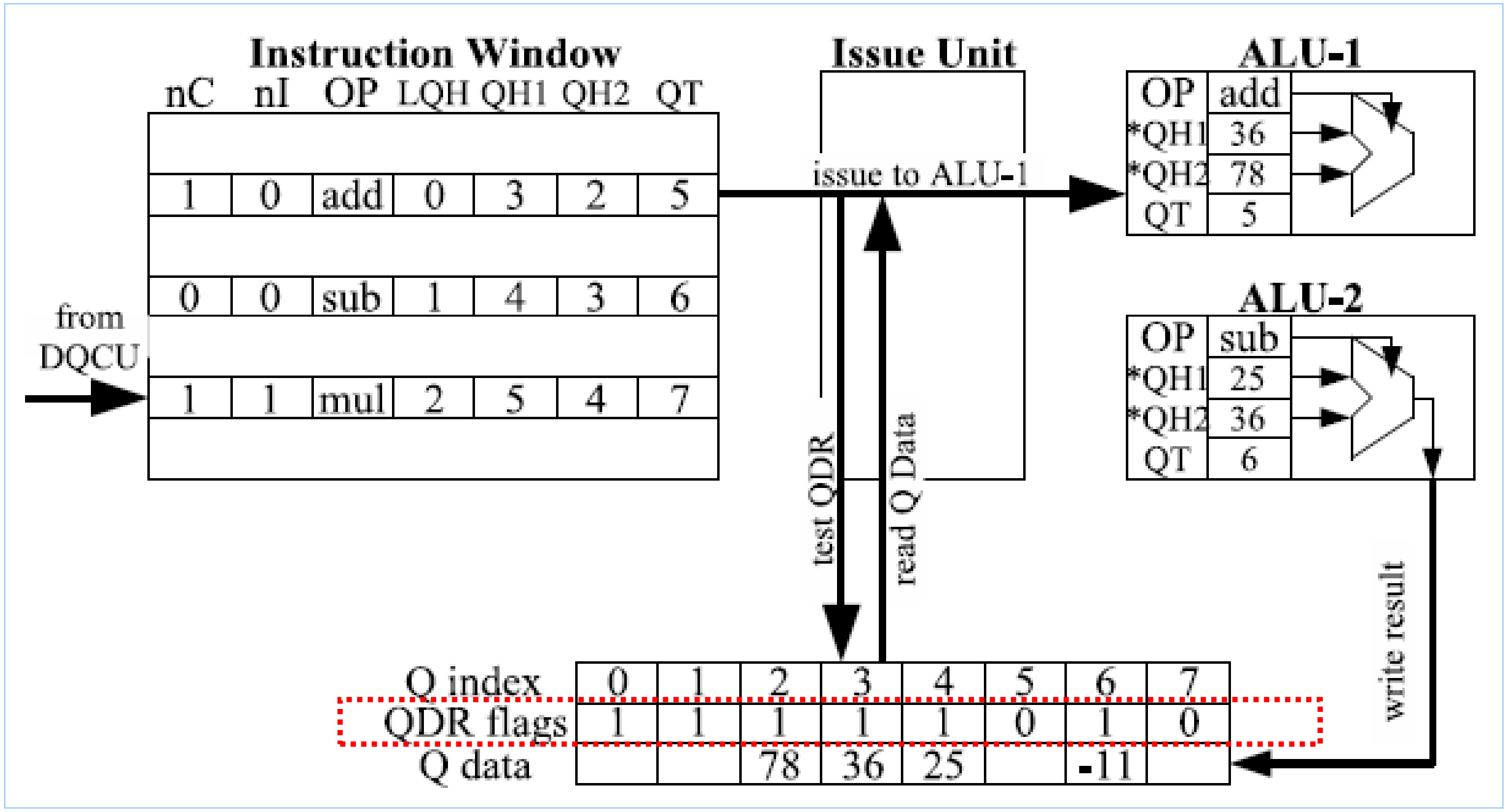
OFFSET: positive/negative integer value that indiactes the location of SRC2(n-1) from the QH(n-1)
QTn : queue tail value of instruction n
DESTn : destination location of instruction n

QueueCore architecture
Queue computation – example



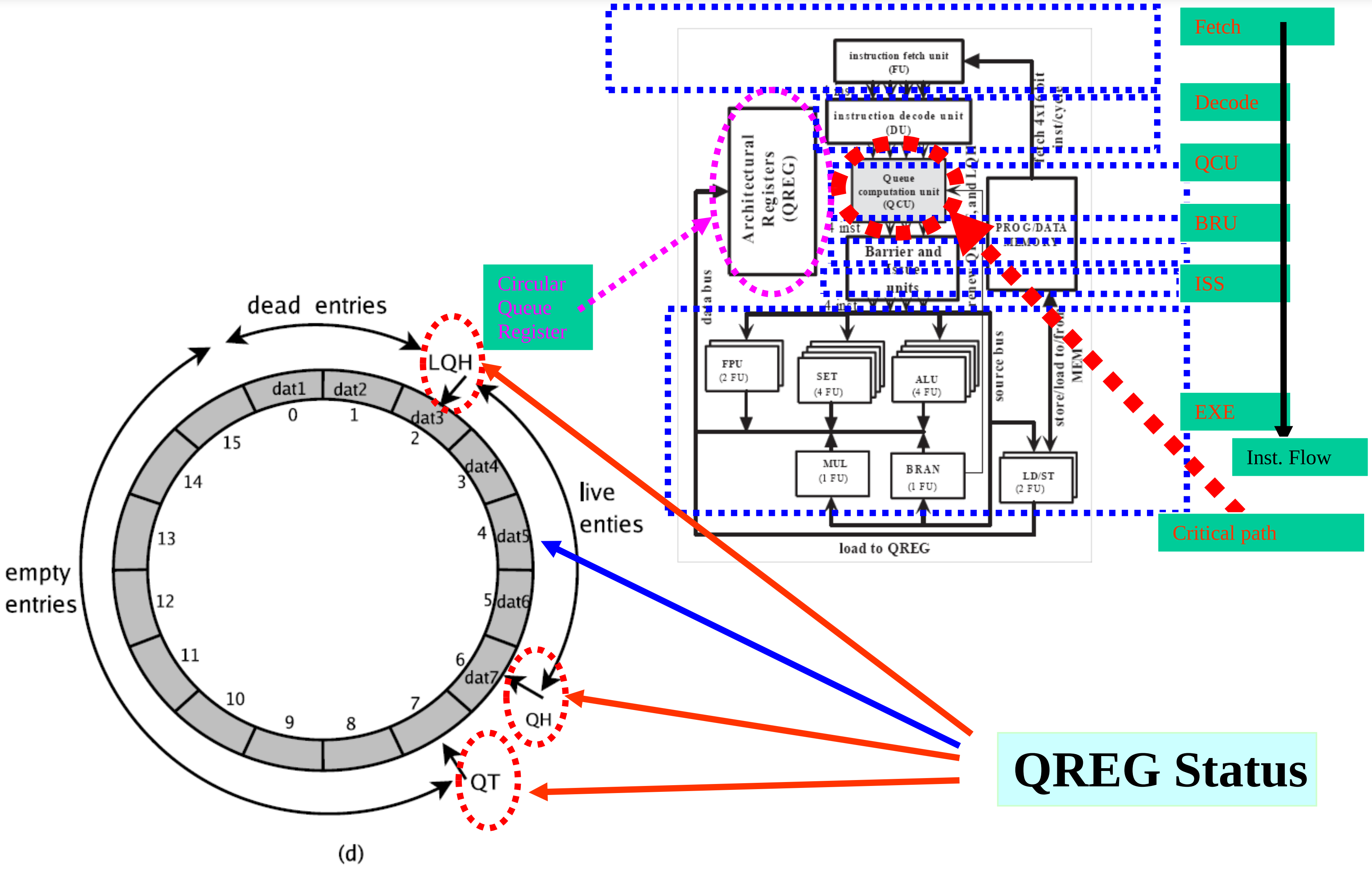
QueueCore architecture

Instructions Issue

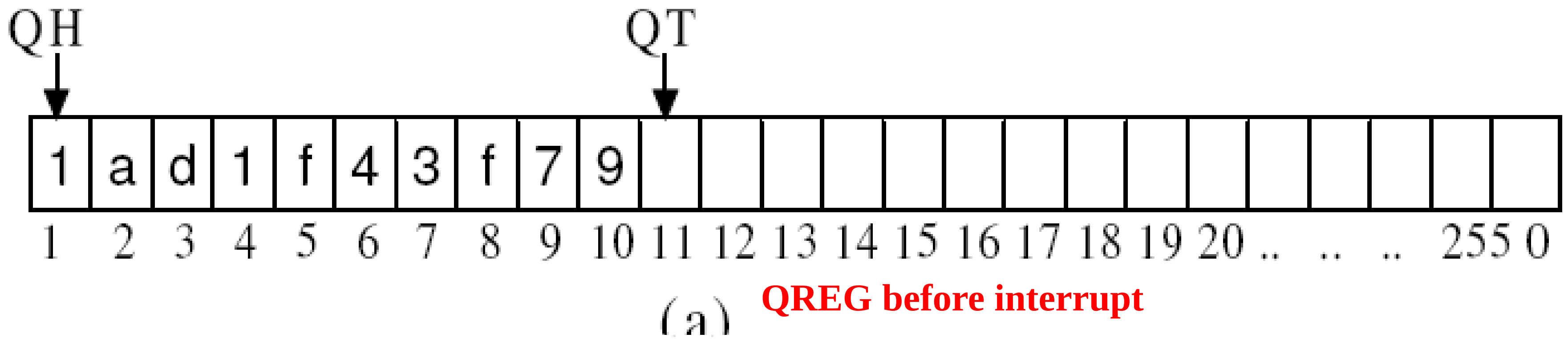


QueueCore architecture

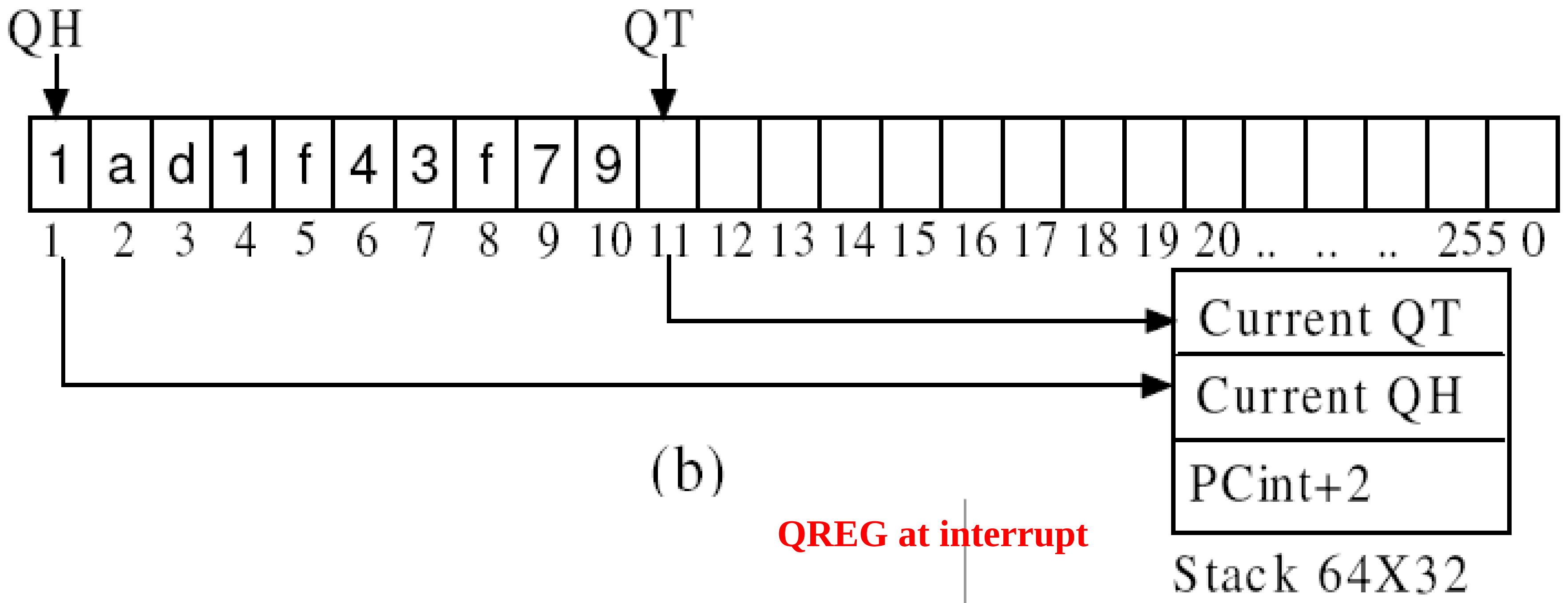
Interrupt handling



QueueCore architecture
Interrupt handling

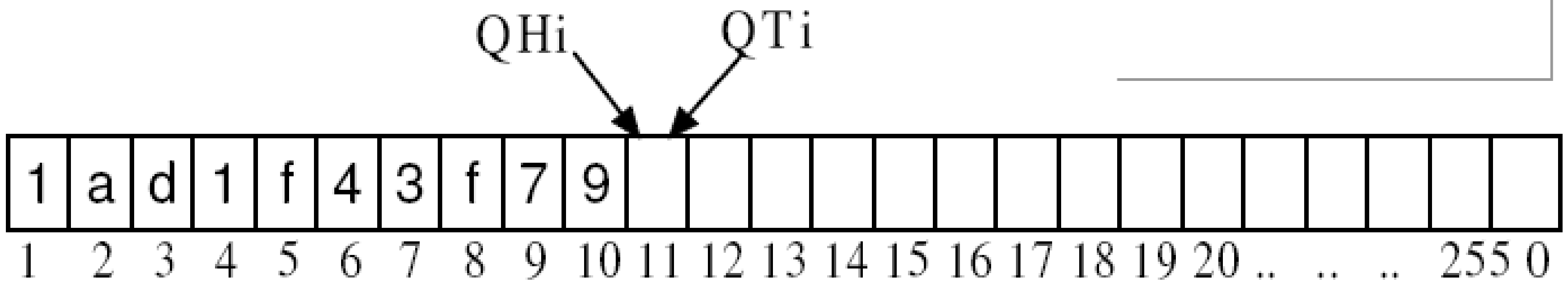


QREG before interrupt

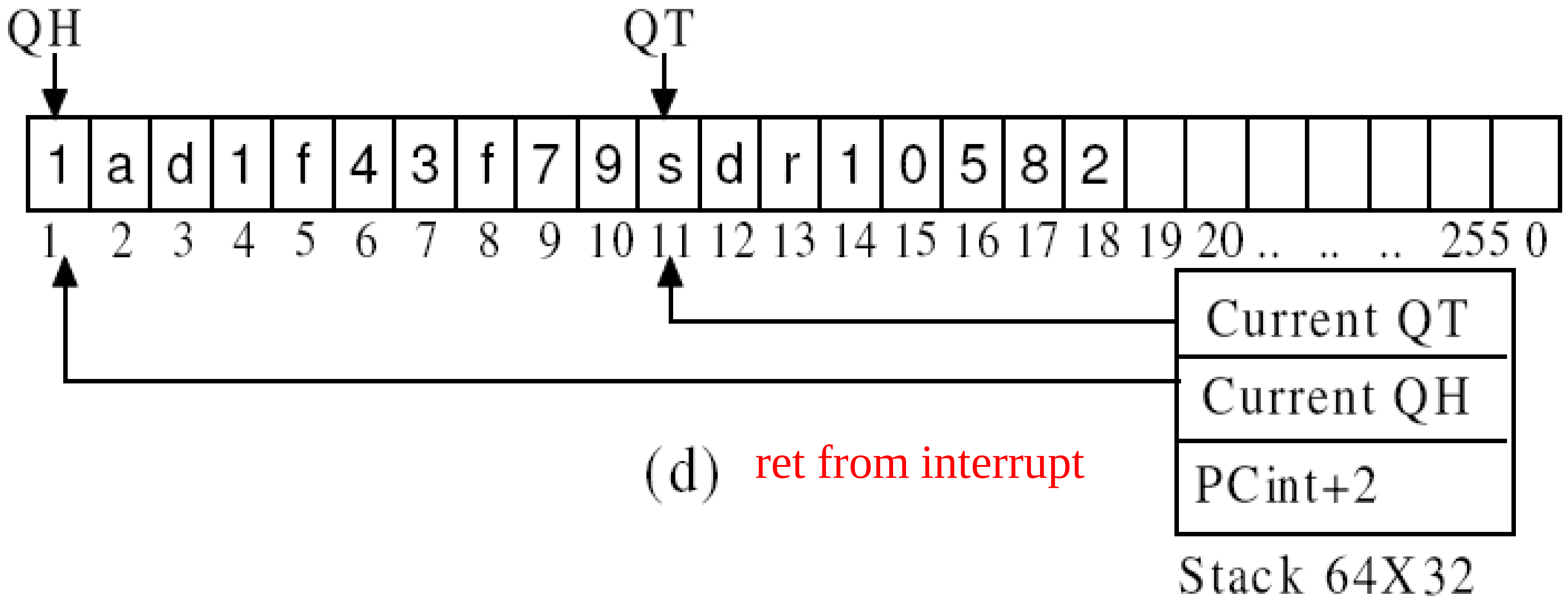


QREG at interrupt

QueueCore architecture
Interrupt handling

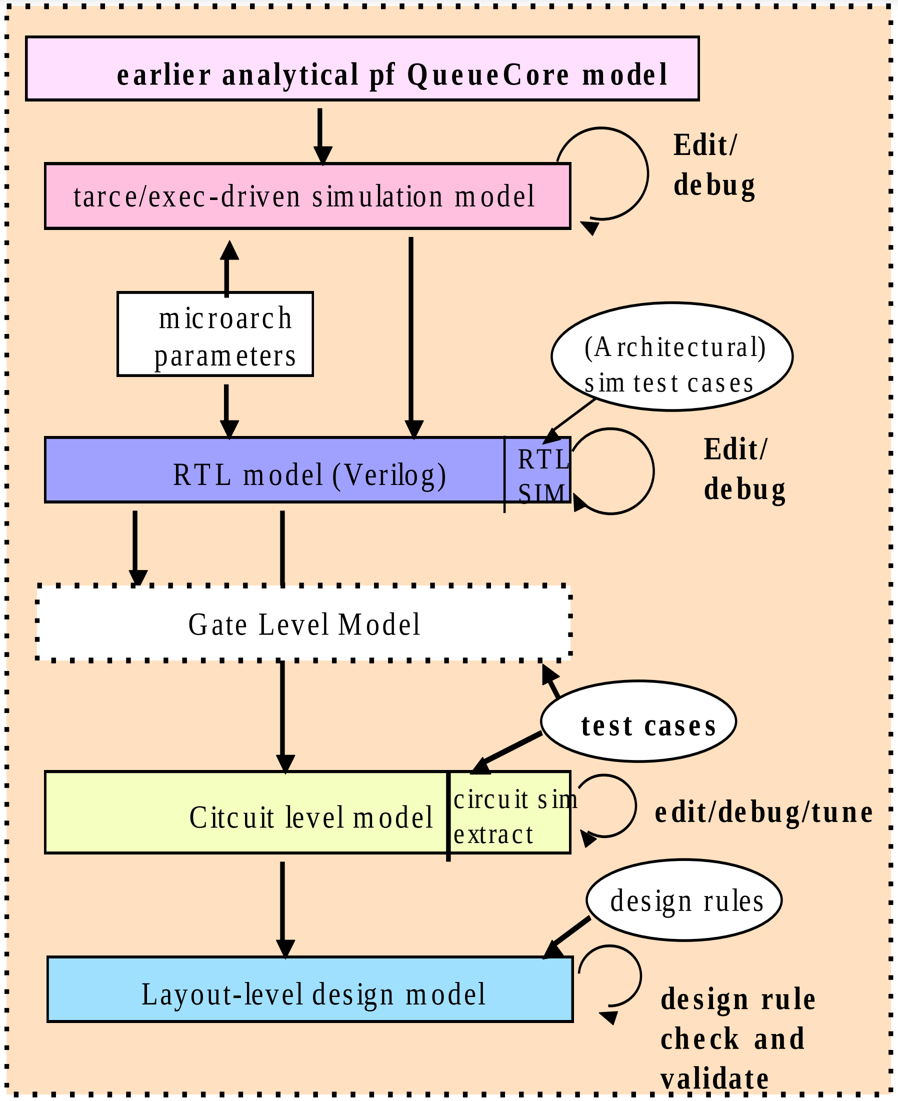


(c) Interrupt Queue allocation

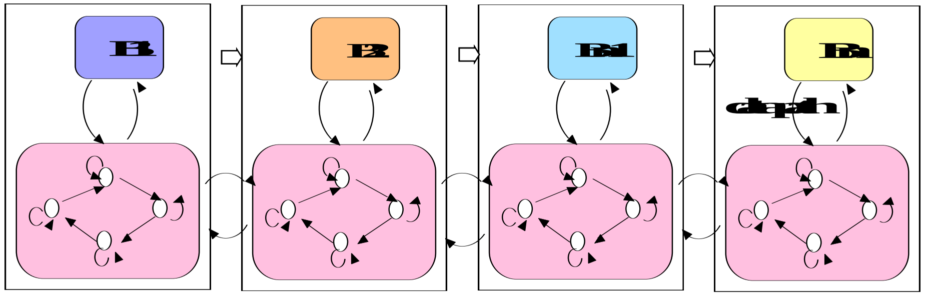


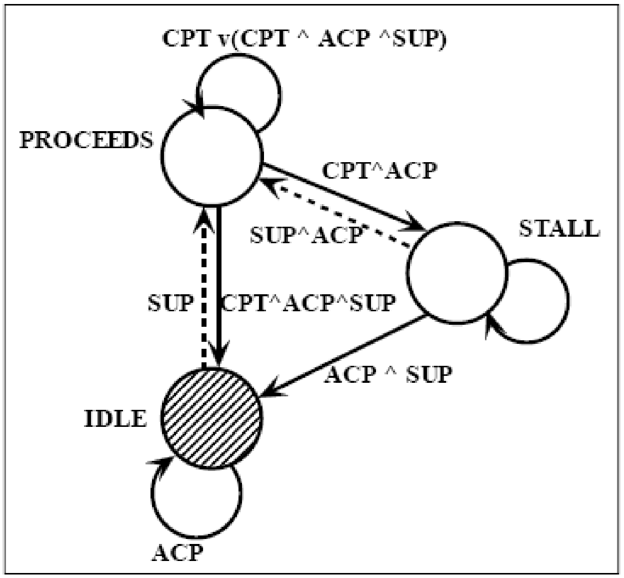
(d) ret from interrupt

QueueCore design Methodology



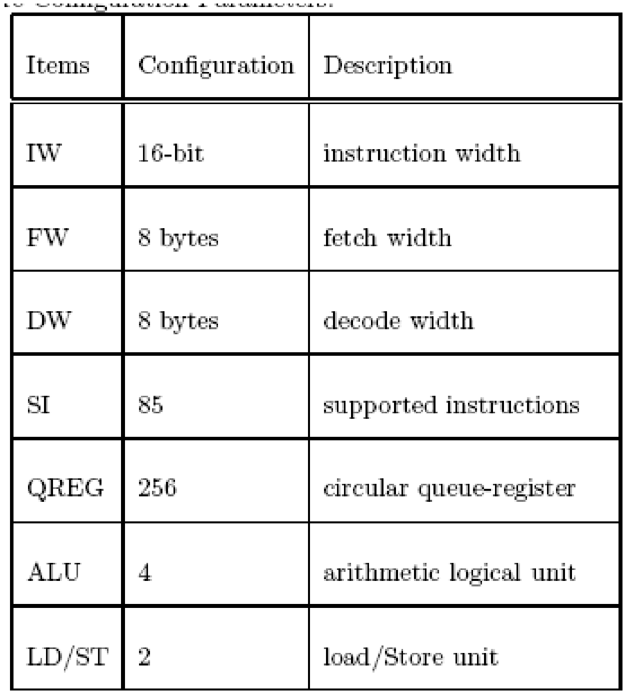
- ➢Modular Design Methodology
- ➢Decentralized control

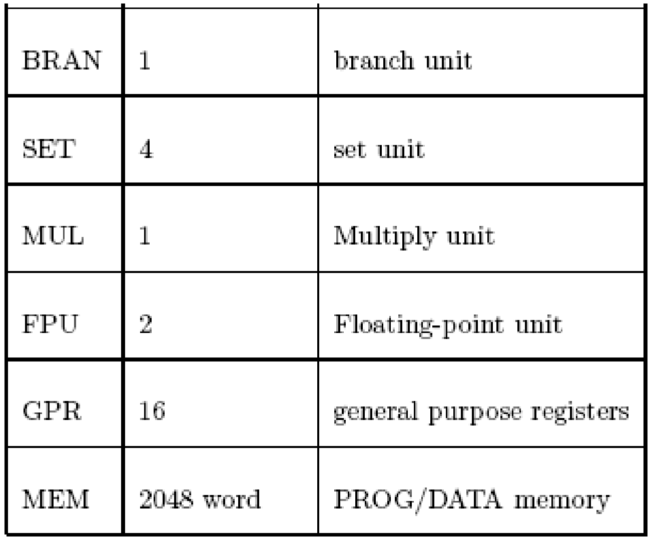




Finite state machine transition for QC-2 pipeline synchronization

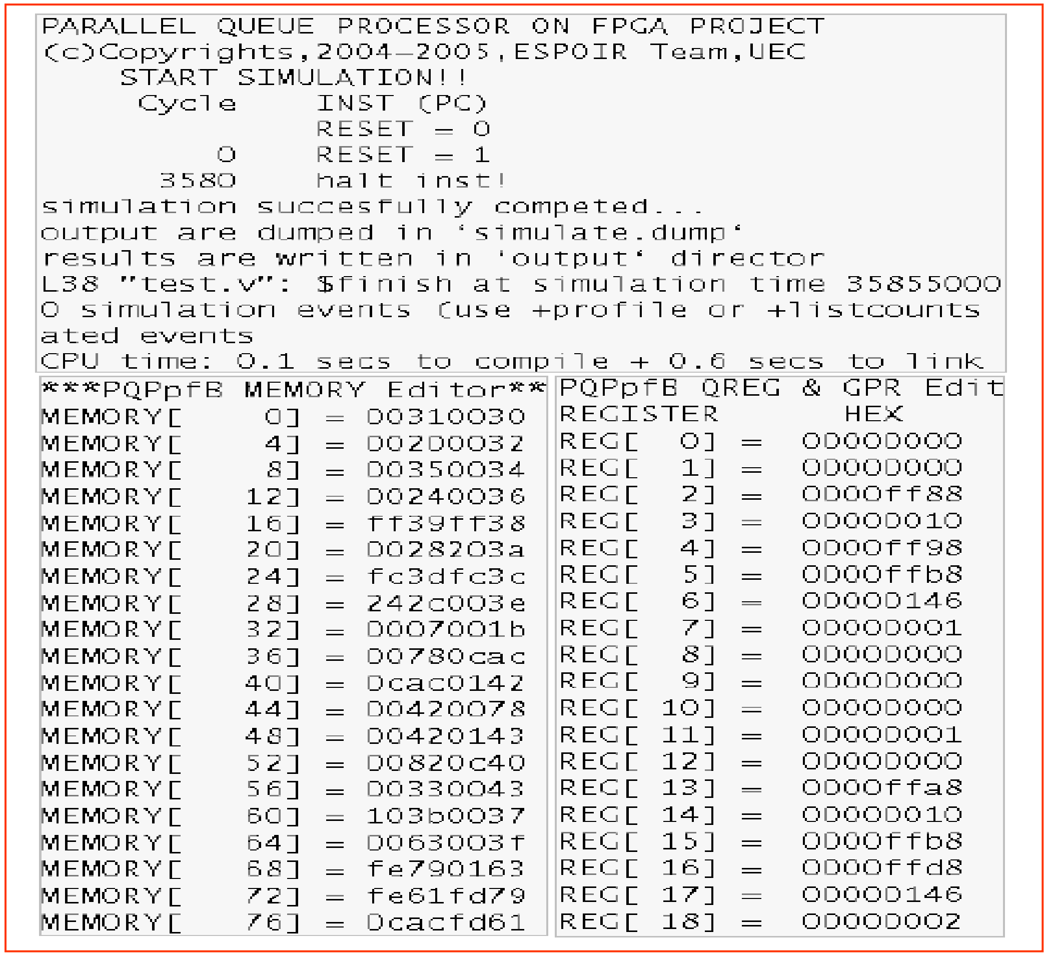
QueueCore design
Hardware configuration and tools





Items	Configuration	Description
IW	16-bit	instruction width
FW	8 bytes	fetch width
DW	8 bytes	decode width
SI	85	supported instructions
QREG	256	circular queue-register
ALU	4	arithmetic logical unit
LD/ST	2	load/Store unit
BRAN	1	branch unit
SET	4	set unit
MUL	1	Multiply unit
FPU	2	Floating-point unit
GPR	16	general purpose registers
MEM	2048 word	PROG/DATA memory

QueueCore design
Verification and Debugging



```
PARALLEL QUEUE PROCESSOR ON FPGA PROJECT
(c)Copyrights,2004-2005,ESPOIR Team,UEC
    START SIMULATION!!
     Cycle     INST (PC)
               RESET = 0
         0     RESET = 1
      3580     halt inst!
simulation succesfully competed...
output are dumped in 'simulate.dump'
results are written in 'output' director
L38 "test.v": $finish at simulation time 35855000
0 simulation events (use +profile or +listcounts
ated events
CPU time: 0.1 secs to compile + 0.6 secs to link
```

```
***PQPpfB MEMORY Editor**
MEMORY[    0] = 00310030
MEMORY[    4] = 00200032
MEMORY[    8] = 00350034
MEMORY[   12] = 00240036
MEMORY[   16] = ff39ff38
MEMORY[   20] = 0028203a
MEMORY[   24] = fc3dfc3c
MEMORY[   28] = 242c003e
MEMORY[   32] = 0007001b
MEMORY[   36] = 00780cac
MEMORY[   40] = 0cac0142
MEMORY[   44] = 00420078
MEMORY[   48] = 00420143
MEMORY[   52] = 00820c40
MEMORY[   56] = 00330043
MEMORY[   60] = 103b0037
MEMORY[   64] = 0063003f
MEMORY[   68] = fe790163
MEMORY[   72] = fe61fd79
MEMORY[   76] = 0cacfd61
```

```
PQPpfB QREG & GPR Edit
REGISTER          HEX
REG[  0]  =  00000000
REG[  1]  =  00000000
REG[  2]  =  0000ff88
REG[  3]  =  00000010
REG[  4]  =  0000ff98
REG[  5]  =  0000ffb8
REG[  6]  =  00000146
REG[  7]  =  00000001
REG[  8]  =  00000000
REG[  9]  =  00000000
REG[ 10]  =  00000000
REG[ 11]  =  00000001
REG[ 12]  =  00000000
REG[ 13]  =  0000ffa8
REG[ 14]  =  00000010
REG[ 15]  =  0000ffb8
REG[ 16]  =  0000ffd8
REG[ 17]  =  00000146
REG[ 18]  =  00000002
```

QueueCore design
Synthesis results

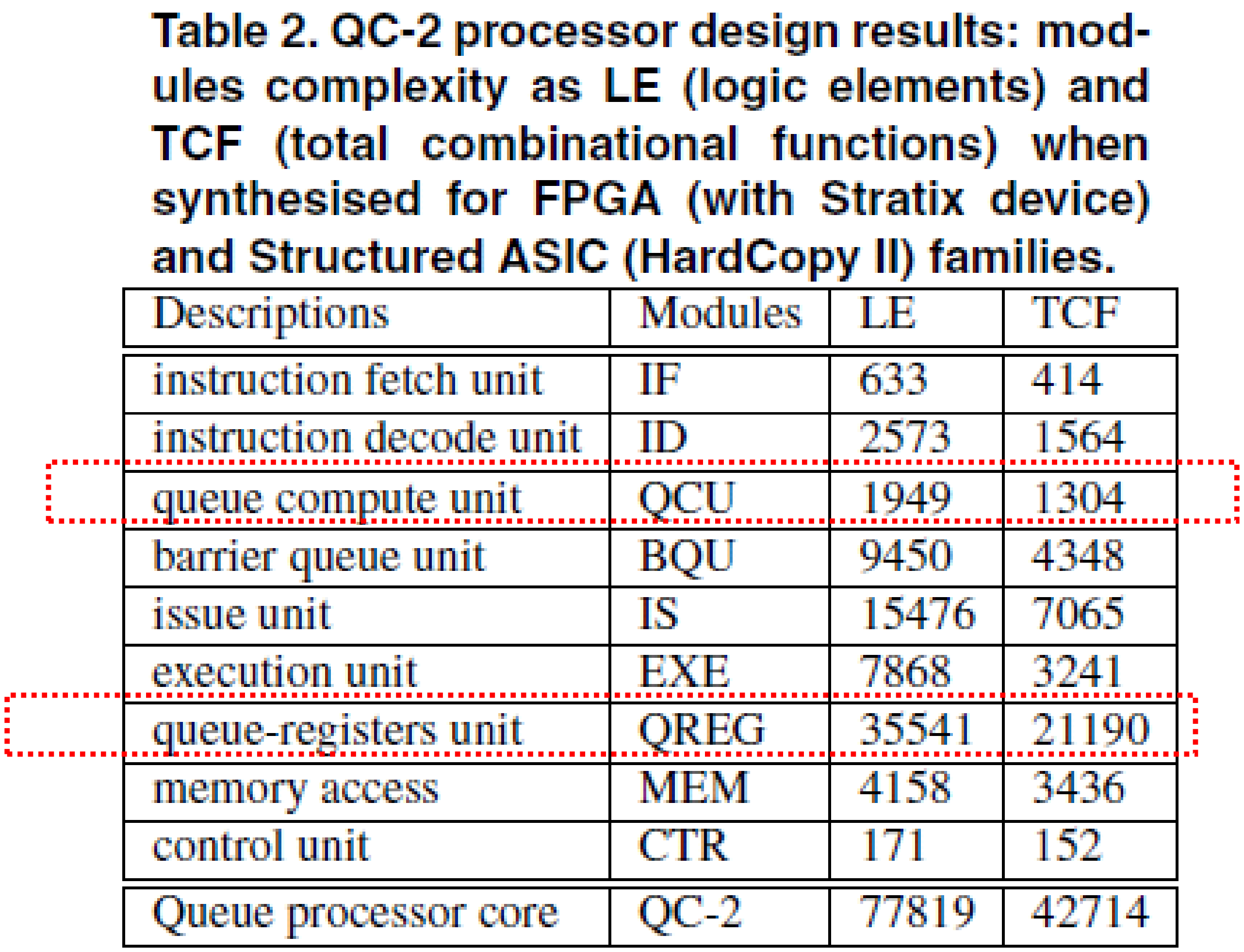
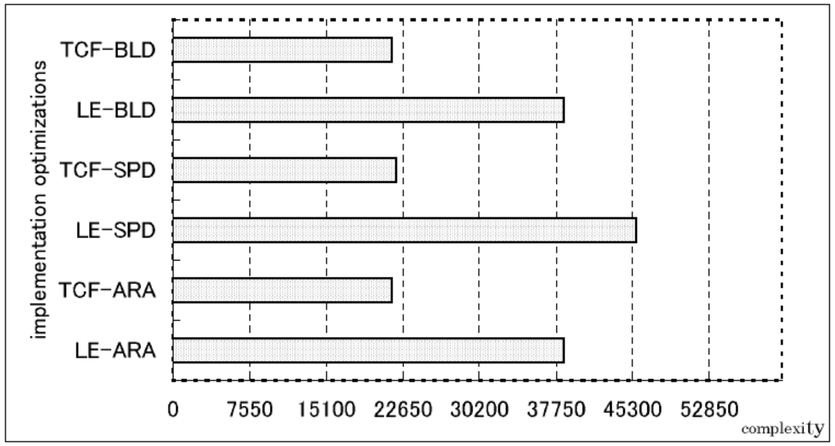


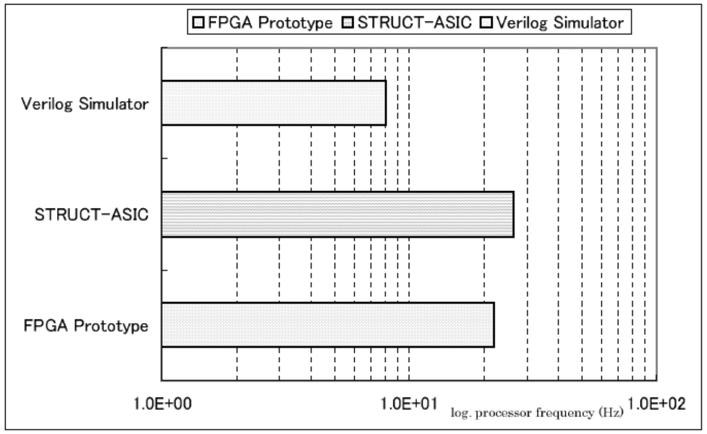
Table 2. QC-2 processor design results: modules complexity as LE (logic elements) and TCF (total combinational functions) when synthesised for FPGA (with Stratix device) and Structured ASIC (HardCopy II) families.

Descriptions	Modules	LE	TCF
instruction fetch unit	IF	633	414
instruction decode unit	ID	2573	1564
queue compute unit	QCU	1949	1304
barrier queue unit	BQU	9450	4348
issue unit	IS	15476	7065
execution unit	EXE	7868	3241
queue-registers unit	QREG	35541	21190
memory access	MEM	4158	3436
control unit	CTR	171	152
Queue processor core	QC-2	77819	42714

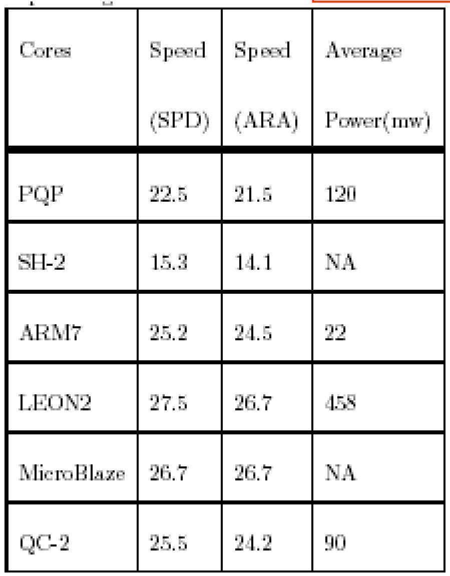
QueueCore design Performance results



Resource usage for 256*33 QREG file



Achievable Frequency (Nominal Frequency rating)



Cores	Speed (SPD)	Speed (ARA)	Average Power(mw)
PQP	22.5	21.5	120
SH-2	15.3	14.1	NA
ARM7	25.2	24.5	22
LEON2	27.5	26.7	458
MicroBlaze	26.7	26.7	NA
QC-2	25.5	24.2	90

Comparison with synthesizable cores