Kalray kv3 V1 VLIW Core Optimization Guide

Kalray S.A.

This document provides some details of the MPPA® architecture, primarily addressed to assembly language programmers and compiler writers

150 pages

Kalray kv3 V1 VLIW Core Optimization Guide

Kalray S.A.[1]

[1] info@kalray.eu

Abstract: This document provides some details of the MPPA® architecture, primarily addressed to assembly language programmers and compiler writers

Keywords:

Contents

1 Introduction

This document details the Kalray kv3 VLIW core pipeline implementation. The information concerns the Kalray kv3 hardware implementation of the application cores (PE) and management core (RM) of the compute clusters in the Kalray MPPA®-80 Coolidge processor.

It should be considered as an addendum to the *Kalray kv3 VLIW Core Architecture Reference Manual* and is primarily addressed to assembly language programmers and compiler writers. As such, it contains detailed instruction scheduling information, as well as explanations about some particularities/irregularities of the kv3 VLIW core implementation and how to exploit/work-around them.

2 Syntax Conventions

The following syntax conventions apply to this document:

Convention	Definition
mat blue	a simple (32-bit) read action/data
bright blue	a double (64-bit) read action/data
mat red	simple (32-bit) result data is ready
bright red	double (64-bit) result data is ready
mat green	a simple (32-bit) write-back action/data
bright green	a double (64-bit) write-back action/data
ALU	the ALU execution unit
`ALU`	an instruction of the kind “ALU”
`add`	the `add` instruction (inside text)

3 Execution Unit Ports and Latencies

3.1 Micro-Architecture Overview

From an instruction/data dispatch point of view, the kv3 VLIW core features 5 primary execution units (EXUs):

- 2 ALUs
- 1 MAU
- 1 LSU
- 1 BCU

Each of them makes use of some read and write register-file ports to retrieve and store data they process. The implementation of the kv3 VLIW core features 4 simple read ports, 6 double read port, 2 simple write ports, 2 double write ports and 1 special double write port. Table 1 shows, for each EXU, which ports are used at every stage of the pipeline, as well as the stage at which the results are ready, with the following conventions:

RPS i	Read Port Simple number i (read 32 bits register from the register file)
RPD i	Read Port Double number i (read 64 bits register pair from the register file)
WPS i	Write Port Simple number i (commit 32 bits in a general register)
WPD i	Write Port Double number i (commit 64 bits in a general register pair)
WPDS	Write Port Double Special (commit 32 bits in system function register)
Ready	Simple Result is ready and made available for bypass to read ports
Ready	Double Result is ready and made available for bypass to read ports

Instructions executing on a particular EXU do not have to use all the EXU read/write ports thoroughly, e.g. a `mulwdl` (executing on the MAU) will only use one half of WPD 0 as it outputs only 32 bits. Furthermore, some instruction variants use immediate operands. Although these operands are read by the EXUs through their read port, they do not involve any register-file reading and cannot lead to any kind of inter instruction dependence, thus having no effect on the pipeline flow (`make` is probably the best example of such instructions as it only has one immediate read operand).

It is also important to distinguish the EXUs and the instructions they execute. Basically, an 'EXU' executes '`EXU`' kind instructions, i.e. ALUs execute `ALU` instructions, LSU executes `LSU` instructions and so on. However, most EXUs have the extra capacity to execute other kinds of instructions in addition to their "native" kind. More specifically:

- ALUs can execute `ALU` instructions + some (simple) `FPU` instructions
- MAU can execute `MAU` instructions + the other (complex) `FPU` instructions + some ("light") `ALU` instructions

- LSU can execute `LSU` instructions + some ("tiny") `ALU` instructions
- BCU can only execute `BCU` instructions

Thus, floating point instructions, listed as "FPU Instructions" in the *Kalray kv3 VLIW Core Architecture Reference Manual*, are split into two categories: those that are implemented in the ALUs execution units and those that are implemented in the FPU part of the MAU execution unit. These instructions share the read/write ports of the ALUs or those of the MAU according to their category.

The MAU and LSU units respectively comprise a LIGHT ALU and a TINY ALU execution path that use their host's read and write ports and support different subsets of the `ALU` instructions. Namely, the double LIGHT ALU contained in the MAU can handle any `ALU(D)_TINY` or `ALU(D)_LITE` instruction (in the sense of the Reservation classes documented in the architecture manual) while the DOUBLE TINY ALU can only execute `ALU(D)_TINY` instructions. Of course, `ALU` instructions sent to one of these double LIGHT/TINY ALUs execute with the same low latency as guaranteed by the main ALU units. Hence their result is ready at E1 as apparent in the table, which is earlier than a typical `MAU`/`LSU` instruction. Similarly, `FPU` instructions executing in the MAU exhibit a longer latency than typical `MAU` instructions.

In Table 1, these different cases are expressed as MAU/`MAU`, MAU/`FPU` and MAU/`ALU` EXUs. Idem for LSU/`LSU` and LSU/`ALU`. No distinction has been made between ALU/`ALU` and ALU/`FPU` as it makes no difference from a read/write date point of view.

Another feature of the kv3 VLIW core is that the two 32-bits main ALUs can be combined to process 64 bits `ALU` instructions. In this case, all the read and write ports of the two ALUs are available to the instruction with the same read/write stages and latencies as indicated for the two separate ALUs.

3.2 Consequences on the Execution Flow

Once an instruction has entered the Ready stage of the EXU it is executing on, or has flowed further down the pipeline, its result may be bypassed to other instructions upstream that need to read it. Before this stage, the result is not available and data-dependent upstream instructions will have to wait (stall). In the particular case of MAU/`FPU` instructions, the result is ready at E4 and may be bypassed at that cycle, just before being written-back (the Ready was not put in the table for "graphical" reasons only but it should be considered there).

Using Table 1, we may determine the number of stall cycles that will take place between two data-dependent instructions following each other in the pipeline in the general case. Such wait cycles are called RAW (for Read-After-Write) stalls and constitute one of the possible causes of stalls in the kv3 VLIW core pipeline.

For instance, two data-dependent `ALU` instructions may follow each other without any stall cycle (be they executed on main ALUs or on a double LIGHT/TINY ALU). Indeed, when the first one reaches E1, its result is ready and may be directly bypassed to the read port (RPS i or RPD i) used by the second one at RR. If these two `ALU` instructions are alone in their respective

EXU	ID	RR	E1	E2	E3	E4
ALU0		RPS 0 RPS 1	Ready		WPS 0	
ALU1		RPS 2 RPS 3	Ready		WPS 1	
MAU/MAU		RPD 0 RPD 1	RPD 2	Ready		WPD 0
MAU/FPU		RPD 0 RPD 1 RPD 2				WPD 0
MAU/ALU		RPD 0 RPD 1	Ready			WPD 0
LSU/LSU		RPD 3 RPD 4	RPD 2	Ready	WPD 1	
LSU/ALU		RPD 3 RPD 4	Ready		WPD 1	
BCU	RPD 5				WPDS	

Table 1: EXUs read/write ports and latencies

bundle, then the execution of the second bundle may follow that of the first one without any stall. For example:

```
add $r0 = $r1, $r2
;;                  ## no stall between these two bundles,
                    ## $r0 is bypassed from E1 to RR
add $r3 = $r0, $r4
;;
```

On the other hand, a branch instruction reading a GPR (a register of the general-purpose register file) needs it at stage ID (through RPD 5). If this GPR is the target of a `MAU` instruction of the preceding bundle, then the branch instruction (and of course the potential other instructions of its bundle) will spend 2 cycles stalling at ID while the `MAU` bundle progresses in the pipeline. When the latter reaches E2, the result of the `MAU` instruction is ready, bypassed to the branch instruction still waiting at ID (through RPD 5) and the branch bundle resumes execution. At this point in time, stages RR and E1 contain no valid bundles (they are “bubbles” or “holes“ in the pipeline), which reflects the fact that we lost two cycles waiting on the RAW dependency.

```
add $r0 = $r1, $r2
mulwdl $r32 = $r31, $r30
;;                      ## 2 cycles stall between these two bundles,
                        ## then $r32 is bypassed from E2 to ID
                        ## One cycle later, $r0 is bypassed from E3 to RR
add $r3 = $r0, $r4
cb.odd $r32, label_1
;;
...
label_1:
```

To compute the number of stall cycles a particular bundle incurs, all the instructions composing it must be analyzed and checked for RAW stalls with respect to all the bundles that are downstream in the pipeline. This is in fact how the hardware proceeds, stalling a bundle as long as necessary at ID, RR or E1 when it needs to read an operand that is not yet available.

```
add $r0 = $r1, $r2
muluuwd $r42r43 = $r0, $r20
lw $r32 = 0[$r30]
;;
sbf $r60 = $r61, $r62    ## no stall for this bundle:
sw 0[$r50] = $r42        ## $r42 is bypassed from E2 to E1 (through RPD2)
;;
xor $r60 = $r60, $r0     ## this bundle stalls 1 cycle at fetch
cb.odd $r32, label_2     ## to let the lw progress to E2
;;                       ## then $r32 is bypassed from E2 to ID
...
label_2:
```

Of course, these stall cycles depend on exactly which operands are used by the concerned instructions and on their sizes. Only true data dependencies trigger stalls. So for example:

```
ffmawd $r32r33 = $r40r41, $r31, $r30  ## MAU/FPU: result ready only at E4
;;
                                ## 3 cycles stall between these two bundles,
                                ## then $r32 is bypassed from E4 to RR

fabs $r3 = $r0, $r32            ## ALU/FPU: needs $r32 at RR so waits 3 cycles there
;;
```

But:

```
ffmaw $r33 = $r40, $r31          ## MAU/FPU: $r33 result Ready only at E4
;;
                                 ## No true data dependency => no stall

fabsw $r3 = $r0, $r32            ## ALU/FPU: needs $r32 at RR
;;
```

In the example above, although WBDP 0 (that is a double write port) is dedicated to the write back of the MAU results in the register file, the `ffma` instruction only uses its top half as it writes only `$r33` that is 32-bit register (and not a register pair). As a result, `$r32` is not seen as targeted by the first bundle in the scoreboard and no RAW stall occurs.

The tables in Section 8 describe exactly which operands of what size are read/written at every stage of the pipeline for all the instructions of the instruction set.

3.3 Instruction Pipeline Particularities

Looking at Table 1, several particularities appear that will help explain some irregularities/constraints of the kv3 VLIW core pipeline.

3.3.1 Shared Double Read Port

First, RPD 2 is shared between the MAU and the LSU. As a result, `LSU` instructions that use this port (mostly stores) cannot be bundled with MAU/* instructions that use it too, as expressed in bundling Rule 9 of Section 7. The instructions using this shared port normally belong to a reservation class of type MAU_ACC* or LSU_ACC*, as stated in Section 9. However, due to a hardware bug in the kv3 VLIW core implementation, no MAU/`FPU` at all can be bundled with a `LSU` instruction using this port. As a result, the only MAU/* instructions that can be bundled with a store/`acws*` are plain MAU/`MAU` instructions, excluding all `mad*/msb*` "MAC" instructions.

Then we see that RPD 2 is marked as used at E1 by MAU/`MAU` and LSU/`LSU`, but RR for MAU/`FPU`. In fact, the real register-file read happens at RR, which allows to give its input data as early as possible to the FPU part of the MAU that has a lot of long processing to do. So if a MAU/`FPU` instruction has a RAW dependency on this port with respect to an other instruction downstream in the pipeline, it will stall at RR until the downstream instruction releases its result . As for MAU/`MAU` instructions and LSU/`LSU` instructions, they make no use of this port before the end of the E1 stage. As a result, in the same scenario they would not be stalled at RR but progress to E1 instead. Once at E1, the situation is re-examined and stalls or bypasses may happen according to the availability of the needed operand at that time. Thus, at the cost of additional hardware and complexity, this mechanism brings a kind of "virtual" E1 double port that is in fact RPD 2 at RR plus bypasses/stalls at E1. The fact that this double port acts as if it were as a E1 port for MAU/`MAU`s instruction and LSU/`LSU` instructions reduces the possibilities of RAW stall, and, in particular, allows to process data-dependent MAC operations at the rate of one per cycle, the E2 result of one being bypassed to the accumulator port of the following at E1.

3.3.2 E3 Write-After-Write Stall

It is also apparent in Table 1 that the write-back port used by the MAU (WPD 0) commits the outputs of this unit in the register-file at the end of E4, be the instruction an MAU/`MAU`, MAU/`FPU` or MAU/`ALU`, whereas all the other write-back ports commit at E3. This has one side effect: if an instruction executing in the MAU is followed on the next cycle by an instruction executing on an other unit (say on an ALU) and they both want to write the same register, then there is a write-after-write conflict (WAW) because they will both want to write the same register at the same time. The hardware deals with it by simply ignoring (squashing) the result of the MAU/`MAU` instruction, only performing the write-back of the instruction in the ALU without any stalls (contrary to the k1a implementation of the core). A subsequent instruction depending (RAW) on the result of the ALU instruction will be un-stalled, as usual, as soon as the ALU instruction reaches E1 (as if the MAU instruction had not been there at all).

In the kv3 VLIW core implementation, the only case when a WAW pseudo-dependency will generate a stall is when an instruction (I) wants to write a GPR (R), that is also targeted by a previously issued streaming load, that is itself still outstanding in the memory system. In this particular case, (I) will be stalled at E3 until the streaming load comes back and commits its result in the register file. Of course, this should not be very common as it requires that no instruction between the streaming load and (I), including (I) itself, reads (R) (otherwise the instruction in question) will classically stall at F, RR or E1 in RAW.

3.3.3 `get` Instruction Bundling

The write-back port of the BCU, WPDS, addresses system function registers (SFRs) such as PS, CS, MMC (see the *Kalray kv3 VLIW Core Architecture Reference Manual* for the full list of them) but it is not a regular write-back port, inasmuch as it cannot target GPRs of the register file.

As a result, the few `BCU` instructions that need to commit a result in the register file use an indirect path, going through the LIGHT ALU of the MAU to gain access to regular write-back port WPD 0. These instructions are:

- `get`, `iget`

Hence, when one of these instructions is issued in a bundle, both the BCU and the MAU are reserved and cannot be used by other instructions of the bundle. Moreover, only 2 tiny `ALU` instructions may be issued in the bundle (that will necessarily execute on ALU0 and ALU1). This is because a third one would be sent to the MAU by the dispatch logic that has no knowledge of the magic that turns a BCU `get` instruction into a MAU instruction in the middle of the pipeline, thus leading to an architecturally undefined result with two instructions trying to execute on the MAU at the same time. However, a regular LSU instruction is allowed in the bundle. These constraints are expressed in bundling Rule 5 of Section 7.

3.3.4 Integer Carry Management

There is one integer carry available in the kv3 VLIW core. It is read and written by `ALU` `addc`, `addcd`, `sbfc` and `sbfcd` instructions and by `MAU` `madsucwd` and `maduucwd` in-

structions; and read only by ALU addci, addcid, sbfci and sbfcid instructions and by MAU madsuciwd and maduuciwd instructions.

As apparent in the tables of Section 8, ALU instructions read it at RR and make it available for bypass (Ready) at E1, and MAU instructions read it at E1 and make it Ready at E2. In particular, this means, for example, that two consecutive addc instructions may execute without stalling, the first one bypassing its output carry to the input of the second one as below:

```
addc $r20 = $r0, $r2
;;
addc $r21 = $r20, $r3  ## no stall: $r20 and the carry are bypassed from E1 to RR
;;
```

The integer carry corresponds to bit 0 of CS (CS.IC) and this is where it is written back at the end of stage E3. set and hfxb instructions targeting CS.IC make their result available for bypass at E1 so that an ALU or MAU instruction needing to read the carry does not stall after them, as stated by Rule 15 of Section 7. Rule 3 of the same Section details bundling constraints related to the carry.

The CS register also holds a carry counter in its bits [31:16]. This is a special hardware counter that auto-increments itself each time a bundle containing an instruction that outputs a carry at 1 reaches E3. Thus it is read-modified-written atomically at E3 and cannot occasion any stalls for "normal" (i.e non BCU) instructions.

3.3.5 IEEE-754 Flags and Rounding Mode

IEEE-754 Flags Some MAU/FPU and some ALU/FPU instructions (see the architectural manual for the list of them) may raise one or more of the 5 IEEE-754 floating point flags:

- invalid operation
- division by zero
- overflow
- underflow
- inexact

These flags are written at E4 in the related bit of the CS register, where they are sticky, i.e. the hardware never clears them (only a software instruction such as set or hfxb can bring them back to zero once they have been raised).
It is valid to bundle together a MAU/FPU instruction and a ALU/FPU instruction that may both raise one or more of these flags. In this case, the flags output by the two execution units are ORed together by pairs at E4 before being written back into the related CS bits.
In any case, the modification of CS is done in an atomic read-modify-write manner at E4 and cannot cause any pipeline stall (except for get, hfx and trapa/o instructions coming after a

`FPU` instruction that could touch these flags).

Instructions that modify these flags are subject to bundling Rule 4 of Section 7.

Rounding Mode Some MAU/`FPU` instructions (see the architectural manual for the list of them) use the RM (Rounding Mode) field of the CS register to apply one or the other rounding type. This field is considered as pseudo-static in the hardware, meaning that no dependency check at all is performed on it: the pipeline will never bypass it nor stall because of it. It is the responsibility of the software to ensure that this field is set to the correct value before issuing any concerned `FPU` instruction to the pipeline. One way to do this is to have a `barrier` instruction follow the instruction that changes the value of CS.RM, as stated in Rule 13 of Section 7.

4 LSU Instructions Latencies

This section covers the latencies of `LSU` instructions executing on the processing (PE) and management (RM) cores of the "compute clusters". The behavior of memory accesses in the quad-RMs of IO-clusters is not the same and will not be detailed in this document, however the main differences can be listed as:

- misalignment is not supported in IO-clusters
- typical load hits return at E3 in IO-clusters (instead of E2 in compute clusters)
- accesses to the SMEM have a longer latency (+2 cycles) in the IO-cluster
- of course accesses to the DDR memory have a much longer latency than accesses to the SMEM

Moreover, it should be noted that all the access latencies (the time that an access takes to return its result to the core) and all the cache busy latencies (the time during which the L1 Data cache is busy and does not grant any access, e.g. after a miss) given in this section are subject to conflicts/arbitration in the memory system (including conflicts between L1 I cache and L1 D cache of the same core) as soon as accesses are not strictly restricted to the core data cache. As a result, and not taking into account maintenance instructions:

- only hits latencies of load instruction are guaranteed
- all misses and uncached accesses, as well as all store instructions, deal with the memory system and/or with the internal state of the write buffer so there is no guarantee on their latency / cache busy time

The cycle figures given for accesses dealing with the memory system correspond to best execution times, when no conflict whatsoever happens and the road to/from the SMEM is free.

In the compute clusters, RMs and PEs access the memory using either cached or uncached accesses. The compute cluster kv3 VLIW core data cache characteristics are as follows:

- cache size: 16KB = 4KB per way * 4 ways
- line size: 64 Bytes
- cached accesses follow a write-through write-around policy with stores going to the SMEM through a write-buffer
- the write buffer contains 8 64-bit words, is fully associative, allows recombination and uses a true LRU eviction policy
- it always tries to keep 1 word free (out of 8), meaning that it will spontaneously try to evict the LRU word when full

4.1 Load Instructions

4.1.1 Cached Load Instructions

Hits In the compute cluster implementation of the kv3 VLIW core, regular cached load accesses that hit make their result available for bypass at the end of stage E2. Thus, reading a load result in the following bundle will result in a 1 cycle stall if the consumer EXU is an ALU, 2 if it is a BCU, 0 if it is a LSU using the result as store data, etc. as may easily be computed using Table 1 of Section 3.

```
lw $r0 = 0[$r20] ## hits
;;
add $r5 = $r0, $r2 ## stalls 1 cycle at RR then $r0 is bypassed from E2 to RR
;;
```

```
lw $r0 = 0[$r20] ## hits
;;
cb.eqz $r0, some_label ## stalls 2 cycles at ID then $r0 is bypassed from E2 to ID
;;
```

```
lw $r0 = 0[$r20] ## hits
;;
sw 10[$r5] = $r0 ## no stalls: $r0 is directly bypassed from E1 to RR
;;
```

Misses On a load miss, the data cache (L1D cache) performs a line refill and will not accept any request before its completion. A typical refill lasts 17 cycles as the SMEM is 10 cycles away and the L1D cache lines are 64 bytes wide (=> 8*64 bits transfers for a refill burst => 10 cycles to get the first 64-bit word + 7 cycles to get the 7 remaining words). This means that in the example code below:

```
lw $r0 = 0[$r20]   ## misses
;;
sd 10[$r5] = $r6r7 ## stalls 17 cycles at E1 as L1D cache is busy
;;
```

the store will stall 17 cycles at E1 and be granted by the L1D cache on the 18th cycle.

In addition to memory system delays, another cause is susceptible to make that figure rise: the miss in L1D cache / hit in write buffer scenario. This happens when a load misses in the cache but the write buffer contains data that belong to the cache line where the load missed. In such a situation, there is a danger of inconsistency as the L1D cache could refill data from the memory system that are not up to date. To prevent such troubles and keep the hardware simple, the whole write buffer (that is 8 x 64 bits words) is flushed to the SMEM first, and then the refill sequence is performed. As a result, the busy time of the cache will be increased by 1 to 8 cycles (according to the actual filling rate of the write buffer), thus reaching 18 to 25 cycles instead of 17.

The hardware implementation uses a critical-word-first scheme that allows to bring back first the words requested by the core from the memory system during refills. Hence, in the above example, the load word instruction will receive its result (at E3 where it has been waiting for it during the miss), commit it in the register file and exit the pipeline before the store is released,

because the loaded data are forwarded by the data cache to the core before the end of the refill. In this particular case, it does not have any interesting effect on the core execution flow as the store will remain stalled at E1 until the end of the refill, blocking the rest of the pipeline upstream anyway. However, critical-word-first is interesting if the result of the load is needed (e.g.) for computation and no other access is made to the L1D cache for some cycles:

```
lw $r0 = 0[$r20]  ## misses
;;
add $r5 = $r0, $r2 ## stalls 11 cycles at RR then $r0 is bypassed from E3 to RR
;;
xor $r5 = $r5, $r30 ## any instructions but loads/stores
sbf $r50 = $r40, 35
mulwdl $r24 = $r51, $r8
;;
add $r50 = $r50, 3 ## any instructions but loads/stores
;;
nop  ## any instructions but loads/stores
;;
nop ## any instructions but loads/stores
;;
nop ## any instructions but loads/stores
;;
lw.add.x1 $r28 = $r29[$r31] ## granted by the D$ at E1 with no stall
;;
```

As precised above, in the kv3 VLIW core implementation, a L1D cache refill is 64 bytes wide and the SMEM memory system typically answers it in 8 consecutive 64 bits words, so the L1D cache is likely to grant again 7 cycles after it transmitted the critical-word to the core. This means that:

- **the pure load miss penalty is 10 cycles** (= execution latency of 11 cycles), to which we must add the usual (hit case) number of RAW dependency stall cycles: 1 for ALUs, 2 for BCU, etc. as seen in the hits subsection.
- **the L1D cache is unavailable for an extra 7 cycles after the critical-word return**, hence a total of 17 stall cycles (= execution latency of 18) if we wanted to access it directly after the load, as in the first example of this subsection.

In the code above, the `add $r5 = $r0, $r2` will stall 10 + 1 = 11 cycles at RR. The `lw.add.x1 $r28 = $r29[$r31]` will proceed without additional stalls, though it is only 6 bundles further than the instruction that uses the critical-word (`add $r5 = $r0, $r2`), because the add has waited at RR (and not E1), thus giving the load an extra cycle to go to E1.

Misalignment Penalties The kv3 VLIW core supports data misalignment, that is a load or a store instruction may target an object whose address is not a multiple of its size, like a `ld` at an address that does not end in 0x0 or 0x8, such as 0x10004 for example. The hardware will execute such accesses correctly but they may incur latency penalties.

As mentioned previously, a L1D cache line width is 64 bytes. As long as a load does not overlap two lines, there is no penalty except if it misses and overlaps 2 64 bits words. In this particular case, the load penalty goes from 10 to 11 as the critical-word-first logic must wait for

two 64 bits words before forwarding the result to the core.

If a load overlaps two lines, as would, for example, a `lw` at address 0x2003e, then the data cache breaks the access in two, performs each half-access in turn (that can hit or miss independently) and finally concatenates the two half results as needed before presenting the data back to the core. The fact that the access is split in two always adds one cycle to the execution time of the load. Another additional cycle is needed to concatenate the two half results if the second half access hits (if it misses, this cycle is present anyway to help achieve a good timing on data returning from the memory so we do not count it as extra).
We may now compute the latencies of misaligned loads that cross L1D cache lines in the four possible hit/miss configurations:

1st half status	2nd half status	1st half latency	2nd half latency	add. misal. penalty	total exec latency	total use penalty	total L1D cache busy
hit	hit	1	1	1	3	2	2
hit	miss	1	11	0	12	11	18
miss	hit	18	1	1	20	19	19
miss	miss	18	11	0	29	28	35

where *total exec latency* represents the total number of cycles taken to execute the load, *total use penalty* represents the number of cycles an instruction immediately following the load and needing the load result at E1 (i.e. with no extra RAW stall) would spend stalling at this stage (so total use penalty = total exec latency - 1) and *total L1D cache busy* represents the number of cycles an instruction immediately following the load and wanting to access the L1D cache at E1 would spend stalling at this stage.
Note that the latency of a first half miss is considered to be 18 (instead of 11 for a regular load miss) because the 2nd half access cannot start as long as the L1D cache is busy refilling the line in which the 1st half missed. By comparison, a regular access (not crossing L1D cache lines) would be represented as such:

access status	access latency	add. misal. penalty	total exec latency	total use penalty	total L1D cache busy
hit	1	0	1	0	0
miss (regular)	11	0	11	10	17
miss (ovlp 2 x 64-bit)	11	1	12	11	18

4.1.2 Uncached Load Instructions

In the kv3 VLIW core, uncached loads come in two flavors: blocking and non-blocking (streaming). A load may be uncached either because the data cache is turned off (PS.DCE = 0), or because it is an explicit uncached instruction (such as `lwu`) or because the MMU decided it was uncached (MMU off and address in the peripheral space, or MMU on and address in an

uncached/device page). Streaming is enabled for uncached loads when PS.USE = 1, as defined in the *Kalray kv3 VLIW Core Architecture Reference Manual*.

On one hand, blocking uncached loads stop the core execution by stalling the load bundle at E3 in wait for its answer. On the other hand, streaming uncached loads (also simply known as "streaming loads" since only uncached loads may be streaming) allow the core to continue executing as soon as the load has successfully departed to the SMEM and as long as the pipeline is not stalled by a RAW dependency on the result of the load. The loaded data will then be committed in the register file asynchronously to the core instructions flow.

The typical execution latency of an uncached blocking load is 10 cycles (1 cycle at E1 + 1 cycle at E2 + 8 cycles stalled at E3), which is equivalent to 9 cycles of *total use penalty* as defined in the previous section, and 9 cycles of *total L1D cache busy*. This is one cycle better than the execution latency of a miss because, as the hardware does not have any hit/miss check to perform, the access departs to the memory system one cycle earlier.
These figures must typically be increased by one if the access spans two 64 bits words.

As for the streaming loads, they also typically exhibit a latency of 10 cycles, equivalent to 9 cycles of *total use penalty*. However, they do not keep the cache busy (once they have been sent to the memory system), so other accesses can be "streamed" to the SMEM through it (cached and uncached blocking accesses can of course be performed too). Below is the table summarizing this information in the usual format:

access type	access latency	add. misal. penalty	total exec latency	total use penalty	total L1D cache busy
uncached blocking	10	0	10	9	9
uncached blocking ovlp	10	1	11	10	10
uncached streaming	10	0	1	9	0
uncached streaming ovlp	10	1	1	10	2

Of course, the meaning of *access latency* is a little different for streaming loads: what it really means here is the typical number of cycles until the result of the streaming load is available. Once again, contrary to blocking loads, streaming loads will not stall the pipeline once they have gone out of the L1D cache unless they are the source of a RAW dependency stall (in which case they typically produce *total use penalty* cycles. The fact that they do not block the pipeline is reflected by the value of 1 for *total exec latency*.

Streaming loads *access latency* is subject to the same increments as blocking loads *access latency* with respect to misalignment. In addition, a misaligned streaming load will cause the L1D cache to appear busy during 2 cycles (*total L1D cache busy* = 2) instead of 0.

In the kv3 VLIW core implementation, the FIFOs holding target GPRs and formatting information about the outstanding streaming loads are sized so that they can absorb the exact number of loads that match the number of cycles needed to access the SMEM (total outstanding capacity

= 10). This provides the benefit that, with a responsive SMEM, streaming loads can be emitted continuously, at the rhythm of one per cycle without any stalls of the core (provided that the consumers have been scheduled far enough to avoid RAW stalls: 10 bundles further than the load they depend on for E1 readers, 11 bundles for RR readers (e.g. `ALU` instructions)). So with a good scheduling, the kv3 VLIW core is able to issue one streaming load double (8 bytes) every cycle without stalling.

4.2 Store Instructions

4.2.1 Cached Store Instructions

For cached stores, the hardware implements a write-through write-around policy where cached stores always go to the memory system through the write-buffer, and also update the cache itself if they hit in it (in case a store misses in the cache, the latter is left untouched and only the write-buffer is updated, which is the definition of a write-through write-around policy).

We should also mention that stores are posted: this means that, once they have been granted by the L1D cache at E1, their bundle will not be stalled as an effect of the store execution in the cache (contrary to cached loads that may stall their bundle, and the pipeline, when they miss). Thus their execution latency is always 1. However, they may make the L1D cache busy, depending on the state of the write-buffer, and that will stall the pipeline should the core want to access the cache.

Regardless of the hit/miss status of a cached store in the cache (that will only change the fact that the cache itself is updated with the store data or not), three different cases can happen:

- when a well-aligned cached store hits in the write buffer (that is it recombines with data already present in the write buffer) there is no penalty and *total L1D cache busy* equals 0.

- When a well-aligned cached store misses in the write buffer but there is free room left in the latter, there is no penalty and *total L1D cache busy* equals 0.

- When a well-aligned cached store misses in the write buffer and there is no room left in it, the write buffer will try to evict one of its data (the LRU) to the memory system to free a slot for the newcomer. As long as it does not succeed, the cache will be considered busy, refusing to grant any new access. This situation can last from 1 to many cycles according to the availability of the memory system. However, this should typically not happen too often as the write buffer always tries to have at least one free slot at any given time.

Misalignment Penalties If a cached store is misaligned but does not overlap 2 64 bits words, there is no misalignment penalty whatsoever.

If it overlaps 2 64 bits words, then there is a misalignment penalty due to the fact that the write buffer is organized as a (write only) fully associative cache of 8 64 bits words. Hence it cannot handle a 64 bits words overlapping access in one cycle. Consequently, the store will be

cut in two halves that will be handled separately in the write buffer. This will always generate one cycle of *L1D cache busy* that should be added to the cycles of *L1D cache busy* generated by each half access, according to its hit/miss status in the write-buffer, as described above.

4.2.2 Uncached Store Instructions

Just as for the loads, uncached stores come in two flavors: blocking and streaming.

Unlike any other kinds of stores, uncached blocking stores are not posted, that is the instruction will wait at E3 until the "store done" notification comes back from the memory system. This allows to get precise traps in case of memory errors (such as DSECCERROR, DDECCERROR or DSYSERROR). Consequently, an uncached blocking store exhibits the same timing characteristics as an uncached blocking load: a typical execution latency of 10 cycles (1 cycle at E1 + 1 cycle at E2 + 8 cycles stalled at E3) and 9 cycles of *total L1D cache busy*.
These figures must typically be increased by one if the access spans two 64 bits words.

Streaming stores, on the other hand, are posted so they do not block the core execution (unless they are not immediately granted by the memory system once at E2 in the L1D cache), neither do they typically keep the L1D cache busy.
However, as for streaming loads, their *total L1D cache busy* is increased by 2 (so going from 0 to 2) if the access spans two 64 bits words.
Contrary to streaming loads, streaming stores do not need to be pushed into any FIFO so their number is virtually unlimited, that is the core has the ability to send as many streaming stores as wanted to the memory system without stalling (unless the SMEM itself stops granting requests).

access type	access latency	add. misal. penalty	total exec latency	total L1D cache busy
uncached blocking	10	0	10	9
uncached blocking ovlp	10	1	11	10
uncached streaming	1	0	1	0
uncached streaming ovlp	1	1	1	2

4.3 Atomic Instructions

There are two variants of each atomic memory access instructions in the kv3 VLIW core ISA: cached and uncached. Cached atomic instructions are `acws`, `afda` and `aldc`. Uncached atomic instructions are `acwsu`, `afdau` and `aldcu`. One important peculiarity to note about `acws`, `afda` and `aldc` is that they do not obey to the cache policy dictated by the MMU: these instructions will always be cached whatever the cache policy found in the TLB entry that maps the virtual page they belong to. (As usual for *U instructions, `acwsu`, `afdau` and `aldcu` also ignore the MMU cache policy directive and are always uncached).

From a latency point of view, the uncached atomic instructions can be thought of as blocking uncached loads that would need one extra cycle to come back, giving a typical execution latency of 11 cycles (1 cycle at E1 + 1 cycle at E2 + 9 cycles stalled at E3), equivalent to 10 cycles of *total use penalty*, and 10 cycles of *total L1D cache busy*.
As for cached atomic instructions, they deliver their result to the core with the same latency as a cached load (incurring the same hit/miss *total use penalty*) but their *total L1D cache busy* is that of a cached load plus that of a cached store plus one.

4.4 Data Cache Maintenance Instructions

`dinval` invalidates all the lines of the L1D cache, which is instantaneous in the hardware implementation as the valid bits are in registers. Its execution latency is 1 and its *total L1D cache busy* time is 0.

`dinvall` invalidates the concerned line if the provided address hits in the cache else does nothing. For the same reason as `dinval`, this instruction has an execution latency of 1 and a *total L1D cache busy* of 0.

`dtouchl` acts as a cached load that would not commit anything in the core's register file. It can be used to pre-heat some addresses in the cache. It has no *total use penalty* (since it does not bring back any data to the core itself) but it does exhibit the *total L1D cache busy* time of a well aligned cached load.
`wpurge` orders to purge the write buffer (i.e.: flush its content to the memory system and invalidate all the words). It is posted so its execution latency is 1. Its *total L1D cache busy* time can be anything between 0 and a lot of cycles according to the write buffer content and the memory system responsiveness, but 7 should be fairly typical.

4.5 Special LSU Instructions

Two `LSU` instructions cannot be categorized as loads nor stores nor maintenance instructions: `dzerol` and `fence`.
`dzerol` acts as 8 consecutive `sd` instructions, filling a L1D cache line entirely with zeros.
When uncached:

- it is always considered streaming (even if PS.USE = 0)
- it will typically stall the core pipeline at E3 for 7 cycles
- it will have a *total L1D cache busy* time of 8 cycles

When cached, it is posted (so will not directly stall the core) but typically exhibit a *total L1D cache busy* of 7 (according to the write buffer state; the write buffer will always split the access in 8)
`fence` is an instruction that stalls at E1 until there are no more data accesses "in flight" in the memory system, i.e. all ongoing accesses must have terminated and their acknowledgment

must have been received by the L1D cache before `fence` can progress to E2 and let another instruction enter E1. It can be used after a `wpurge` instruction, or after a `sdu` instruction for example to make sure that the memory has been updated before executing other `LSU` instructions. Its execution latency is entirely dependent on the dynamic context of the memory system at the point of its entrance at E1.

4.6 Instruction Cache Maintenance Instructions

`iinval` invalidates all the icache lines instantaneously (execution latency: 1 cycle, total L1D cache busy: 0)

`iinvall` invalidates one icache set instantaneously (execution latency: 1 cycle, total L1D cache busy: 0)

5 BCU Instruction Particularities

5.1 Branches

Branch instructions are divided in two categories: those that branch at ID and those that branch at RR.
The instructions that branch at ID incur a 1-cycle branch penalty because the PFB is flushed and a new request to the L1I cache is made at the branch address (if everything goes well, a new bundle will be ready to execute 2 cycles later). They are:

- `break`
- `call`
- `goto`
- `ret`
- `rfe`
- `scall`
- `trap*`
- `loopgtz` under certain conditions
- `loopnez` under certain conditions

`loopnez` and `loopgtz` branch at ID if the hardware loop start condition is not fulfilled: the loop body must then be skipped, hence the branch.

The instructions that branch at RR incur a 2-cycles branch penalty because the PFB is flushed and a new request to the L1I cache is made at the branch address plus the ID stage is canceled. They are:

- `cb`
- `icall`
- `igoto`

5.2 Pipeline Flushes

Some `BCU` instructions need to have a clean pipeline downstream to start executing. They will stall at ID stage until there is no valid bundle left at RR, E1, E2, E3 and E4 (though there might still be pending streaming loads in the streaming loads FIFO). Such instructions may remain stalled at ID as little as 0 cycle (if the pipeline is already free in front of them), or as long as several hundred cycles (e.g. if the pipeline is full of unlucky load misses that are going to be unfavorably arbitrated in the memory system). However, long stalls coming from load misses

were going to happen and block the pipeline anyway so it is not really fair to count them in this manner. For a typical case, these instructions are responsible for an extra 5 cycles stall (the time that the bundle that is at RR when they first enter ID executes and leaves the pipeline). These 5 cycles of stall come in addition to the potential branch penalty of the instruction, as is the case for `rfe` for example. These instructions are:

- `writetlb`
- `readtlb`
- `probetlb`
- `indexjtlb`
- `indexltlb`
- `barrier`
- `idle*`
- `break`
- `rfe`
- `scall`
- `trap*`
- `hfx*/set` on PS
- `loop*`

In addition, `idle*` and `barrier` instructions wait until:

- L1I cache / memory system traffic is over;
- L1D cache / memory system traffic is over;
- and the streaming load FIFO is empty.

Furthermore, `scall` and `loop*` cannot stall 0 cycle, they always stall at least 1 cycle at ID (for hardware optimization reasons).

5.3 Re-Fetches

Some `BCU` instructions trigger a refetch behind them: that is they branch to the next PC, which has the effect of flushing the PFB and causes an additional 1 cycle stall. This is the case of:

- `barrier`
- `writetlb`

- `invalitlb`
- `hfx*/set` on PS
- `loop*` under certain conditions

Hardware loop instructions (`loop*`) trigger a refetch when their loop body (pcoff17 = LE - `loop*` PC, as defined in the *Kalray kv3 VLIW Core Architecture Reference Manual*) is strictly smaller than 64 bytes (because the PFB might have fetched ahead beyond the loop end as it was not yet aware that we were in a hardware loop).

6 Prefetch Buffer Effects

The prefetch buffer (PFB) is the unit that is in charge of fetching the code in the instruction cache (L1I cache) and presenting the syllables to the pipeline at the output of its FIFOs. The ID stage, using the data from the PFB FIFOs will in turn form the bundles, dispatch the instructions to the decoders of the related EXUs and perform part of the instruction decoding. If ID is not stalled and a valid bundle could be formed, then the corresponding code syllables will be popped out of the PFB FIFO and the bundle will progress to RR on the next clock edge.

The PFB has no conscience of what a bundle is (or what instructions are) and its requests to the L1I cache do not have any correlation with bundle borders. Simplifying a little, its interface with the L1I cache consists of a simple request/grant → answer protocol plus address and data signals.

In the first phase of this protocol, the PFB asks for 128bits of data at a 32-bit aligned address (these "data" are code in fact but, from the PFB point of view, it really looks like raw data). When the L1I cache grants the request (handshake), the PFB can present a new request, either continuing sequentially (so that address 2 = address 1 + 16 bytes) or branching to a new address, for example on order of the BCU.

In the second phase, the L1I cache answers in one or several cycles, indicating each time which parts of its return data bus are valid (1 valid bit per 32-bit word). The PFB must accept the data coming back. As a matter of fact, it will push every valid 32-bit word sent by the L1I cache into one of its 4 FIFOs (according to their alignment).

The PFB features 4 32-bit wide / 3 stages deep FIFOs that hold the code syllables before they are popped out and progress down the pipeline as a bundle. In the nominal case where the L1I cache hits and no L1I cache line is crossed, the PFB receives the full 128-bit of data it asked for on the cycle following the request (valid bits = 0xf). If the bundles are 16-byte wide (= 128 bits = 4 * 32 bits = 4 * 1-syllable instructions), the PFB will ask 128 bits per cycle to the L1I cache (L1I cache stage1), receive 128 bits per cycle from it and push them into its FIFOs (L1I cache stage2), and the pipeline will pop 128 bits per cycle out of the FIFOs. Thus we reach a steady state with a throughput of 4 * 32-bit syllables of code fetched by cycle. This nominal case is illustrated in the figure below where each cross represents a 32 bits syllable

L1I cache st 1 (handshake)	**L1I cache st 2** (answer)	**PFB FIFOs**			**ID**	**RR**
	x			x		
request 4	x push 4			x	pop 4	
⟹	x ⟹			x	⟹	
	x			x		

6.1 L1I cache Line Crossing Penalty

When the kv3 VLIW core branches, the PFB FIFOs are flushed (because the PFB usually fetched some code ahead of time in a linear progression and this code is not wanted anymore) and a new 128 bits request is made to the L1I cache at the branch address. This address is 32 bits aligned but it does not have to be 128 bits (16 bytes) aligned. As a result, if the branch address is not 16-byte aligned, given that the PFB fetches 16 bytes at a time, an L1I cache line is going to be crossed sooner or later, meaning the 16 bytes requested by the PFB will overlap a L1I cache line border.

When this happens (and it is bound to happen quite quickly in the absence of a second branch as L1I cache lines are 64-byte wide), the L1I cache will only return the 32 bits syllables that reside in the same line as the start address of the request. For example if the PFB requests 128 bits starting at address 0x10034 and the L1I cache hits, then the L1I cache will answer at the next cycle with valid bits at 0xe, indicating that it could only provide 32-bits syllables lying at 0x10034, 0x10038 and 0x1003C, the remaining one being beyond the L1I cache line border. It is then the responsibility of the PFB to issue a new request starting at the beginning of the new line (0x10040 in our example).

The PFB request address (distinct from the PC) thus realigns itself on 128-bits and the situation is back to normal, however there was a cycle during which we only retrieved 96 bits instead of the usual 128 bits. So we "lost the occasion" to fetch one syllable. This loss can go up to 3 syllables in case the branch is made to an address ending in 0xc. This could in turn lead to a bubble in the pipeline if we were not capable to feed it with code (starvation).

The following series of figure illustrates the case where a ID branch (such as a `call`) is made to the address 0x23C. Here is an example of code that leads to such a situation:

```
make $r0 = 0xa
sw 0[$r12] = $r34
call my_func    ## branch at ID
;;
my_func:        ## this is a 4 syllables bundle
add $r0 = $r0, $r1
xor $r3 = $r4, $r5
sbf $r40 = $r34, $r38
lw $r32 = 0[$r2]
;;
```

State of the L1I cache/PFB at cycle 0 (cycle of the branch): PFB FIFOs are flushed, 4 syllables are requested to the L1I cache at 0x23C.

L1I cache st 1 (handshake)	**L1I cache st 2** (answer)	**PFB FIFOs**			**ID**	**RR**
request 4	~~x~~			x	pop 1	
@ 0x23C	~~x~~ flush !			x	$\Longrightarrow$	
$\Longrightarrow$	~~x~~ $\Longrightarrow$			x	call !	make
	~~x~~			x		sw

State of the L1I cache/PFB at cycle 1: the L1I cache only gives us 1 syllable back, ID wants

to pop 4 syllables but 0 are available, so ID is starved.

L1I cache st 1 (handshake)	L1I cache st 2 (answer)	PFB FIFOs	ID	RR
request 4 @ 0x240 ⟹	push 1 ⟹ x		starved	call

State of the L1I cache/PFB at cycle 2: the PFB FIFOs contain only 1 syllable but ID wants to pop 4, so ID is starved, bubble at RR.

L1I cache st 1 (handshake)	L1I cache st 2 (answer)	PFB FIFOs	ID	RR
request 4 @ 0x250 ⟹	x push 4 ⟹ x	x	starved	bubble

State of the L1I cache/PFB at cycle 3: the PFB FIFOs contain 5 syllables. ID can finally execute and 4 syllables will be popped out of the PFB FIFOs at the end of the cycle.

L1I cache st 1 (handshake)	L1I cache st 2 (answer)	PFB FIFOs	ID	RR
request 4 @ 0x260 ⟹	x push 4 ⟹ x	x / x / x / x x	pop 4 ⟹	bubble

With this sequence, we have created 2 holes in the pipeline instead of one usually for ID branches. The L1I cache line crossing penalty made us waste one cycle (indeed we could have resumed execution at cycle 2 if we had been able to fetch 4 syllables as usual instead of one).

6.2 Optimization Rules

Always align the beginning (that is the first bundle of the loop body, not the `loop*` instruction) of your hardware loops on a 16-byte boundary: if there are no branches inside the loop body, this will prevent a fetch request from overlapping two L1I cache lines, thus avoiding the L1I cache line crossing penalties. This is not hard to do and may gain a fair amount of cycles, depending on the size of the loop body and the bundling inside it.

7 Instruction Constraints

7.0.1 Instruction Bundling Constraints

Rule 1 Two data dependent instructions may not be scheduled within a single bundle.

Rule 2 Two instructions of the same bundle must not target the same architectural resource, otherwise the update value of this resource is unpredictable.

Rule 3 (Specialization of Rule 2) An ALU instruction that produces a carry must not be bundled with a `SET` or `WFXL` instruction on CS.

Rule 4 (Specialization of Rule 2) An ALU or MAU instruction that produces an IEEE 754 floating point flag must not be bundled with a `SET` or `WFX*` instruction on CS.

Rule 5 If a `GET`, `IGET` or `WAITIT` instruction is issued within a bundle, then the BCU and the tiny ALU unit inside the MAU are used. As a consequence:

- No MAU instruction is allowed in this bundle
- Only 2 tiny ALU instructions may be used in this bundle (one on each ALU).

This rule is implicit and covered by the instruction reservation tables.

7.0.2 Instruction Scheduling Constraints

Rule 6 – Mandatory PC must not be the target of a `hfx*` or a `set` instruction, else a TRAP.OPCODE is generated. A simple branch may be used to modify the PC if needed.

Rule 7 – Mandatory The following instructions may only be issued alone in the bundle:

- `RFE`
- `SCALL`
- `AWAIT`
- `SLEEP`
- `STOP`
- `BARRIER`
- `LOOPDO`
- `TLBREAD`
- `TLBWRITE`

- TLBIINVAL
- TLBDINVAL
- TLBPROBE
- SET/WFXL on PS/MMC/*OW

Rule 8 – Mandatory LS, LE and LC must not be modified within the hardware loop.

Rule 9 – Mandatory The instructions that enable or disable hardware loops must not be used inside the corresponding loop.

Rule 10 – Mandatory To avoid any semantic ambiguity and simplify the design of the hardware, branching instructions, including those triggering implicit refetches, are not allowed at a hardware loop last bundle (generates OPCODE traps). Those are:

- CALL
- ICALL
- CB
- GOTO
- IGOTO
- RET
- RFE
- SCALL
- BARRIER
- LOOPDO
- TLBREAD
- TLBWRITE
- TLBIINVAL
- TLBDINVAL
- TLBPROBE
- SET/WFXL on PS

Rule 11 – Mandatory In order to ensure a correct behavior while executing an hardware loop, a re-fetch must be generated after a set instruction involving LS, LE or LC registers. This re-fetch can be done through a `BARRIER` instruction, but also through the RFE or RET instructions.

Rule 12 – Mandatory Similarly, a refetch has to be generated after an `IINVAL*` instruction. This re-fetch can be done through a `BARRIER` instruction, but also through the `RFE` or RET instructions.

Rule 13 – Mandatory After changing the value of the rounding mode (RM) field of CS, a `BARRIER` instruction must be issued before any MAU/FPU instruction, to guarantee that these instructions will see the new value of RM.

Rule 14 – Mandatory Similarly, after changing the value of PMC, a `BARRIER` is needed before a `GET` instruction on the concerned registers among PM0, PM1, PM2 and PM3, to make sure that the correct value will be read.

Rule 15 – Performance For hardware simplicity, there are no bypasses on system function registers. Then, in case of RAW dependency, the core will stall until the write completion. However, two specific cases are optimized:

- A `RET` instruction after a RA update will be executed seeing immediately the new RA value, without stalls.
- An instruction which involves the carry flag after a CS update will be executed seeing immediately the new carry value, without stalls.

8 Detailed operands read/write stages tables for instructions

The tables of this section show at which stage the (register) read operands of an instruction are read and at which stage its results are ready. The immediate operands, having no effect whatsoever on the pipeline flow (except taking the place of a register operand that could have stalled), are only mentionned in the syntax of the instruction but not in the pipeline part of the tables. The color code used respects the syntax conventions defined in section 2. In particular:

(r) regZ	the simple (32b) register regZ is read
(rd) regP	the double (64b) register regP (a.k.a. pair P) is read
(w) regW	the simple (32b) result that will be written back into regW is ready for bypass
(wd) regM	the double (64b) result data that will be written back into regM is ready for bypass

All the instructions of the instruction set have been grouped in four tables, according to the kind of EXU they execute on. Inside each table, instructions are sorted alphabetically. Section 9 contains an alphabetical index of all the instructions, mentionning their associated EXU (as well as simplified reservation class) and linking to their description in the following tables.

When an alu instruction is sent to a double LIGHT/TINY ALU unit, it exhibits exactly the same behaviour as if it were executed on a big ALU.

Compared to the architecture manual, some names have been shortened (e.g. register → reg, systemPS → PS, etc.) and some complicated immediate names have been simplified (e.g. upper22L_lower10 → imm32) for the sake of readability.

8.1 ALU instructions

Instruction	ID	RR	E1	E2	E3	E4
abdd regW = regZ, extend27_upper27_lower10		(r) regZ	(w) regW			
abdd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
abdd regW = regZ, s10		(r) regZ	(w) regW			
abdd regW = regZ, upper27_lower10		(r) regZ	(w) regW			
abdd regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
abdhq regW = regZ, regY		(r) regZ (r) regY	(w) regW			
abdhq regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
abdw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
abdw regW = regZ, s10		(r) regZ	(w) regW			
abdw regW = regZ, upper27_lower10		(r) regZ	(w) regW			
abdwp regW = regZ, regY		(r) regZ (r) regY	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
abdwp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
absd regW = regZ		(r) regZ	(w) regW			
abshq regW = regZ		(r) regZ	(w) regW			
absw regW = regZ		(r) regZ	(w) regW			
abswp regW = regZ		(r) regZ	(w) regW			
addcd regW = regZ, regY		(r) CS (r) regZ (r) regY	(w) CS (w) regW			
addcd regW = regZ, upper27_lower5		(r) CS (r) regZ	(w) CS (w) regW			
addcd.*i* regW = regZ, regY		(r) CS (r) regZ (r) regY	(w) CS (w) regW			
addcd.*i* regW = regZ, upper27_lower5		(r) CS (r) regZ	(w) CS (w) regW			

Instruction	ID	RR	E1	E2	E3	E4
addd regW = regZ, extend27_upper27_lower10		(r) regZ	(w) regW			
addd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addd regW = regZ, s10		(r) regZ	(w) regW			
addd regW = regZ, upper27_lower10		(r) regZ	(w) regW			
addd regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
addhcp.*c* regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addhcp.*c* regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
addhq regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addhq regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
addsd regW = regZ, extend27_upper27_lower10		(r) regZ	(w) regW			
addsd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addsd regW = regZ, s10		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
addsd regW = regZ, upper27_lower10		(r) regZ	(w) regW			
addshq regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addshq regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
addsw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addsw regW = regZ, upper27_lower5		(r) regZ	(w) regW			
addswp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addswp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
adduwd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
adduwd regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
addw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addw regW = regZ, s10		(r) regZ	(w) regW			
addw regW = regZ, upper27_lower10		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
addwc.*c* regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addwc.*c* regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
addwd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addwd regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
addwp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addwp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
addx16d regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addx16d regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
addx16hq regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addx16hq regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
addx16uwd regW = regZ, regY		(r) regZ (r) regY	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
addx16uwd regW = regZ, upper27_lower5		(r) regZ	(w) regW			
addx16w regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addx16w regW = regZ, upper27_lower5		(r) regZ	(w) regW			
addx16wd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addx16wd regW = regZ, upper27_lower5		(r) regZ	(w) regW			
addx16wp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addx16wp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
addx2d regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addx2d regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
addx2hq regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addx2hq regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
addx2uwd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addx2uwd regW = regZ, upper27_lower5		(r) regZ	(w) regW			
addx2w regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addx2w regW = regZ, upper27_lower5		(r) regZ	(w) regW			
addx2wd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addx2wd regW = regZ, upper27_lower5		(r) regZ	(w) regW			
addx2wp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addx2wp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
addx4d regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addx4d regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
addx4hq regW = regZ, regY		(r) regZ (r) regY	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
addx4hq regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
addx4uwd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addx4uwd regW = regZ, upper27_lower5		(r) regZ	(w) regW			
addx4w regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addx4w regW = regZ, upper27_lower5		(r) regZ	(w) regW			
addx4wd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addx4wd regW = regZ, upper27_lower5		(r) regZ	(w) regW			
addx4wp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addx4wp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
addx8d regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addx8d regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
addx8hq regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addx8hq regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
addx8uwd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addx8uwd regW = regZ, upper27_lower5		(r) regZ	(w) regW			
addx8w regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addx8w regW = regZ, upper27_lower5		(r) regZ	(w) regW			
addx8wd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addx8wd regW = regZ, upper27_lower5		(r) regZ	(w) regW			
addx8wp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
addx8wp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
andd regW = regZ, extend27_upper27_lower10		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
andd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
andd regW = regZ, s10		(r) regZ	(w) regW			
andd regW = regZ, upper27_lower10		(r) regZ	(w) regW			
andd regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
andnd regW = regZ, extend27_upper27_lower10		(r) regZ	(w) regW			
andnd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
andnd regW = regZ, s10		(r) regZ	(w) regW			
andnd regW = regZ, upper27_lower10		(r) regZ	(w) regW			
andnd regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
andnw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
andnw regW = regZ, s10		(r) regZ	(w) regW			
andnw regW = regZ, upper27_lower10		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
andw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
andw regW = regZ, s10		(r) regZ	(w) regW			
andw regW = regZ, upper27_lower10		(r) regZ	(w) regW			
avghq regW = regZ, regY		(r) regZ (r) regY	(w) regW			
avghq regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
avgrhq regW = regZ, regY		(r) regZ (r) regY	(w) regW			
avgrhq regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
avgruhq regW = regZ, regY		(r) regZ (r) regY	(w) regW			
avgruhq regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
avgruw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
avgruw regW = regZ, upper27_lower5		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
avgruwp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
avgruwp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
avgrw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
avgrw regW = regZ, upper27_lower5		(r) regZ	(w) regW			
avgrwp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
avgrwp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
avguhq regW = regZ, regY		(r) regZ (r) regY	(w) regW			
avguhq regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
avguw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
avguw regW = regZ, upper27_lower5		(r) regZ	(w) regW			
avguwp regW = regZ, regY		(r) regZ (r) regY	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
avguwp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
avgw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
avgw regW = regZ, upper27_lower5		(r) regZ	(w) regW			
avgwp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
avgwp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
cbsd regW = regZ		(r) regZ	(w) regW			
cbsw regW = regZ		(r) regZ	(w) regW			
cbswp regW = regZ		(r) regZ	(w) regW			
clrf regW = regZ, stopbit2_stopbit4, startbit		(r) regZ	(w) regW			
clsd regW = regZ		(r) regZ	(w) regW			
clsw regW = regZ		(r) regZ	(w) regW			
clswp regW = regZ		(r) regZ	(w) regW			
clzd regW = regZ		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
clzw regW = regZ		(r) regZ	(w) regW			
clzwp regW = regZ		(r) regZ	(w) regW			
cmovedscalarcond regZ? regW = extend27_upper27_lower10		(r) regZ	(w) regW			
cmovedscalarcond regZ? regW = regY		(r) regZ (r) regY	(w) regW			
cmovedscalarcond regZ? regW = s10		(r) regZ	(w) regW			
cmovedscalarcond regZ? regW = upper27_lower10		(r) regZ	(w) regW			
cmovehqsimdcond regZ? regW = regY		(r) regZ (r) regY	(w) regW			
cmovehqsimdcond regZ? regW = upper27_lower5splat32		(r) regZ	(w) regW			
cmovewpsimdcond regZ? regW = regY		(r) regZ (r) regY	(w) regW			
cmovewpsimdcond regZ? regW = upper27_lower5splat32		(r) regZ	(w) regW			
compdcomp regW = regZ, extend27_upper27_lower10		(r) regZ	(w) regW			
compdcomp regW = regZ, regY		(r) regZ (r) regY	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
compdcomp regW = regZ, s10		(r) regZ	(w) regW			
compdcomp regW = regZ, upper27_lower10		(r) regZ	(w) regW			
compnhqcomp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
compnhqcomp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
compnwpcomp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
compnwpcomp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
compuwdcomp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
compuwdcomp regW = regZ, upper27_lower5		(r) regZ	(w) regW			
compwcomp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
compwcomp regW = regZ, upper27_lower5		(r) regZ	(w) regW			
compwdcomp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
compwdcomp regW = regZ, upper27_lower5		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
copyd regW = regZ		(r) regZ	(w) regW			
copyw regW = regZ		(r) regZ	(w) regW			
ctzd regW = regZ		(r) regZ	(w) regW			
ctzw regW = regZ		(r) regZ	(w) regW			
ctzwp regW = regZ		(r) regZ	(w) regW			
eord regW = regZ, extend27_upper27_lower10		(r) regZ	(w) regW			
eord regW = regZ, regY		(r) regZ (r) regY	(w) regW			
eord regW = regZ, s10		(r) regZ	(w) regW			
eord regW = regZ, upper27_lower10		(r) regZ	(w) regW			
eord regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
eorw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
eorw regW = regZ, s10		(r) regZ	(w) regW			
eorw regW = regZ, upper27_lower10		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
extfs regW = regZ, stopbit2_stopbit4, startbit		(r) regZ	(w) regW			
extfz regW = regZ, stopbit2_stopbit4, startbit		(r) regZ	(w) regW			
fabsd regW = regZ		(r) regZ	(w) regW			
fabshq regW = regZ		(r) regZ	(w) regW			
fabsw regW = regZ		(r) regZ	(w) regW			
fabswp regW = regZ		(r) regZ	(w) regW			
fcdivdsilent2 regW = regP		(rd) regP	(w) regW			(w) CS
fcdivwsilent2 regW = regP		(rd) regP	(w) regW			(w) CS
fcdivwpsilent2 regW = regP		(rd) regP	(w) regW			(w) CS
fcompdfloatcomp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
fcompdfloatcomp regW = regZ, upper27_lower5		(r) regZ	(w) regW			
fcompnhqfloatcomp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
fcompnhqfloatcomp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
fcompnwpfloatcomp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
fcompnwpfloatcomp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
fcompwfloatcomp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
fcompwfloatcomp regW = regZ, upper27_lower5		(r) regZ	(w) regW			
fmaxd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
fmaxhq regW = regZ, regY		(r) regZ (r) regY	(w) regW			
fmaxw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
fmaxwp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
fmind regW = regZ, regY		(r) regZ (r) regY	(w) regW			
fminhq regW = regZ, regY		(r) regZ (r) regY	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
fminw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
fminwp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
fnarrowdwrounding2silent2 regW = regZ		(r) CS (r) regZ	(w) regW			(w) CS
fnarrowdwprounding2silent2 regW = regP		(r) CS (rd) regP	(w) regW			(w) CS
fnarrowwhrounding2silent2 regW = regZ		(r) CS (r) regZ	(w) regW			(w) CS
fnarrowwhqrounding2silent2 regW = regP		(r) CS (rd) regP	(w) regW			(w) CS
fnegd regW = regZ		(r) regZ	(w) regW			
fneghq regW = regZ		(r) regZ	(w) regW			
fnegw regW = regZ		(r) regZ	(w) regW			
fnegwp regW = regZ		(r) regZ	(w) regW			
frecwrounding2silent2 regW = regZ		(r) CS (r) regZ				(w) CS

Instruction	ID	RR	E1	E2	E3	E4
frsrwrounding2silent2 regW = regZ		(r) CS (r) regZ				(w) CS
fsdivdsilent2 regW = regP		(rd) regP	(w) regW			(w) CS
fsdivwsilent2 regW = regP		(rd) regP	(w) regW			(w) CS
fsdivwpsilent2 regW = regP		(rd) regP	(w) regW			(w) CS
fsrecdsilent2 regW = regZ		(r) regZ	(w) regW			(w) CS
fsrecwsilent2 regW = regZ		(r) regZ	(w) regW			(w) CS
fsrecwpsilent2 regW = regZ		(r) regZ	(w) regW			(w) CS
fsrsrd regW = regZ		(r) regZ	(w) regW			
fsrsrw regW = regZ		(r) regZ	(w) regW			
fsrsrwp regW = regZ		(r) regZ	(w) regW			
fwidenlhwsilent2 regW = regZ		(r) regZ	(w) regW			(w) CS
fwidenlhwpsilent2 regW = regZ		(r) regZ	(w) regW			(w) CS
fwidenlwdsilent2 regW = regZ		(r) regZ	(w) regW			(w) CS
fwidenmhwsilent2 regW = regZ		(r) regZ	(w) regW			(w) CS

Instruction	ID	RR	E1	E2	E3	E4
fwidenmhwpsilent2 regW = regZ		(r) regZ	(w) regW			(w) CS
fwidenmwdsilent2 regW = regZ		(r) regZ	(w) regW			(w) CS
insf regW = regZ, stopbit2_stopbit4, startbit		(r) regW (r) regZ	(w) regW			
iord regW = regZ, extend27_upper27_lower10		(r) regZ	(w) regW			
iord regW = regZ, regY		(r) regZ (r) regY	(w) regW			
iord regW = regZ, s10		(r) regZ	(w) regW			
iord regW = regZ, upper27_lower10		(r) regZ	(w) regW			
iord regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
iornd regW = regZ, extend27_upper27_lower10		(r) regZ	(w) regW			
iornd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
iornd regW = regZ, s10		(r) regZ	(w) regW			
iornd regW = regZ, upper27_lower10		(r) regZ	(w) regW			
iornd regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
iornw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
iornw regW = regZ, s10		(r) regZ	(w) regW			
iornw regW = regZ, upper27_lower10		(r) regZ	(w) regW			
iorw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
iorw regW = regZ, s10		(r) regZ	(w) regW			
iorw regW = regZ, upper27_lower10		(r) regZ	(w) regW			
landd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
landd regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
landhq regW = regZ, regY		(r) regZ (r) regY	(w) regW			
landhq regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
landw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
landw regW = regZ, upper27_lower5		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
landwp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
landwp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
liord regW = regZ, regY		(r) regZ (r) regY	(w) regW			
liord regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
liorhq regW = regZ, regY		(r) regZ (r) regY	(w) regW			
liorhq regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
liorw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
liorw regW = regZ, upper27_lower5		(r) regZ	(w) regW			
liorwp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
liorwp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
lnandd regW = regZ, regY		(r) regZ (r) regY	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
lnandd regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
lnandhq regW = regZ, regY		(r) regZ (r) regY	(w) regW			
lnandhq regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
lnandw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
lnandw regW = regZ, upper27_lower5		(r) regZ	(w) regW			
lnandwp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
lnandwp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
lniord regW = regZ, regY		(r) regZ (r) regY	(w) regW			
lniord regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
lniorhq regW = regZ, regY		(r) regZ (r) regY	(w) regW			
lniorhq regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
lniorw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
lniorw regW = regZ, upper27_lower5		(r) regZ	(w) regW			
lniorwp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
lniorwp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
lnord regW = regZ, regY		(r) regZ (r) regY	(w) regW			
lnord regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
lnorhq regW = regZ, regY		(r) regZ (r) regY	(w) regW			
lnorhq regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
lnorw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
lnorw regW = regZ, upper27_lower5		(r) regZ	(w) regW			
lnorwp regW = regZ, regY		(r) regZ (r) regY	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
lnorwp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
lord regW = regZ, regY		(r) regZ (r) regY	(w) regW			
lord regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
lorhq regW = regZ, regY		(r) regZ (r) regY	(w) regW			
lorhq regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
lorw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
lorw regW = regZ, upper27_lower5		(r) regZ	(w) regW			
lorwp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
lorwp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
make regW = extend27_upper27_lower10			(w) regW			
make regW = extend6_upper27_lower10			(w) regW			
make regW = s16			(w) regW			
maxd regW = regZ, extend27_upper27_lower10		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
maxd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
maxd regW = regZ, s10		(r) regZ	(w) regW			
maxd regW = regZ, upper27_lower10		(r) regZ	(w) regW			
maxd regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
maxhq regW = regZ, regY		(r) regZ (r) regY	(w) regW			
maxhq regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
maxud regW = regZ, extend27_upper27_lower10		(r) regZ	(w) regW			
maxud regW = regZ, regY		(r) regZ (r) regY	(w) regW			
maxud regW = regZ, s10		(r) regZ	(w) regW			
maxud regW = regZ, upper27_lower10		(r) regZ	(w) regW			
maxud regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
maxuhq regW = regZ, regY		(r) regZ (r) regY	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
maxuhq regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
maxuw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
maxuw regW = regZ, s10		(r) regZ	(w) regW			
maxuw regW = regZ, upper27_lower10		(r) regZ	(w) regW			
maxuwp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
maxuwp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
maxw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
maxw regW = regZ, s10		(r) regZ	(w) regW			
maxw regW = regZ, upper27_lower10		(r) regZ	(w) regW			
maxwp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
maxwp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
mind regW = regZ, extend27_upper27_lower10		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
mind regW = regZ, regY		(r) regZ (r) regY	(w) regW			
mind regW = regZ, s10		(r) regZ	(w) regW			
mind regW = regZ, upper27_lower10		(r) regZ	(w) regW			
mind regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
minhq regW = regZ, regY		(r) regZ (r) regY	(w) regW			
minhq regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
minud regW = regZ, extend27_upper27_lower10		(r) regZ	(w) regW			
minud regW = regZ, regY		(r) regZ (r) regY	(w) regW			
minud regW = regZ, s10		(r) regZ	(w) regW			
minud regW = regZ, upper27_lower10		(r) regZ	(w) regW			
minud regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
minuhq regW = regZ, regY		(r) regZ (r) regY	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
minuhq regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
minuw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
minuw regW = regZ, s10		(r) regZ	(w) regW			
minuw regW = regZ, upper27_lower10		(r) regZ	(w) regW			
minuwp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
minuwp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
minw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
minw regW = regZ, s10		(r) regZ	(w) regW			
minw regW = regZ, upper27_lower10		(r) regZ	(w) regW			
minwp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
minwp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
movetq regAE = regZ, regY		(r) regZ (r) regY	(w) CS (w) regAE			
movetq regAO = regZ, regY		(r) regZ (r) regY	(w) CS (w) regAO			
nandd regW = regZ, extend27_upper27_lower10		(r) regZ	(w) regW			
nandd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
nandd regW = regZ, s10		(r) regZ	(w) regW			
nandd regW = regZ, upper27_lower10		(r) regZ	(w) regW			
nandd regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
nandw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
nandw regW = regZ, s10		(r) regZ	(w) regW			
nandw regW = regZ, upper27_lower10		(r) regZ	(w) regW			
negd regW = regZ		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
neghq regW = regZ		(r) regZ	(w) regW			
negw regW = regZ		(r) regZ	(w) regW			
negwp regW = regZ		(r) regZ	(w) regW			
neord regW = regZ, extend27_upper27_lower10		(r) regZ	(w) regW			
neord regW = regZ, regY		(r) regZ (r) regY	(w) regW			
neord regW = regZ, s10		(r) regZ	(w) regW			
neord regW = regZ, upper27_lower10		(r) regZ	(w) regW			
neord regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
neorw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
neorw regW = regZ, s10		(r) regZ	(w) regW			
neorw regW = regZ, upper27_lower10		(r) regZ	(w) regW			
niord regW = regZ, extend27_upper27_lower10		(r) regZ	(w) regW			
niord regW = regZ, regY		(r) regZ (r) regY	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
niord regW = regZ, s10		(r) regZ	(w) regW			
niord regW = regZ, upper27_lower10		(r) regZ	(w) regW			
niord regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
niorw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
niorw regW = regZ, s10		(r) regZ	(w) regW			
niorw regW = regZ, upper27_lower10		(r) regZ	(w) regW			
nop						
nord regW = regZ, extend27_upper27_lower10		(r) regZ	(w) regW			
nord regW = regZ, regY		(r) regZ (r) regY	(w) regW			
nord regW = regZ, s10		(r) regZ	(w) regW			
nord regW = regZ, upper27_lower10		(r) regZ	(w) regW			
nord regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
norw regW = regZ, regY		(r) regZ (r) regY	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
norw regW = regZ, s10		(r) regZ	(w) regW			
norw regW = regZ, upper27_lower10		(r) regZ	(w) regW			
notd regW = regZ		(r) regZ	(w) regW			
notw regW = regZ		(r) regZ	(w) regW			
nxord regW = regZ, extend27_upper27_lower10		(r) regZ	(w) regW			
nxord regW = regZ, regY		(r) regZ (r) regY	(w) regW			
nxord regW = regZ, s10		(r) regZ	(w) regW			
nxord regW = regZ, upper27_lower10		(r) regZ	(w) regW			
nxord regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
nxorw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
nxorw regW = regZ, s10		(r) regZ	(w) regW			
nxorw regW = regZ, upper27_lower10		(r) regZ	(w) regW			
ord regW = regZ, extend27_upper27_lower10		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
ord regW = regZ, regY		(r) regZ (r) regY	(w) regW			
ord regW = regZ, s10		(r) regZ	(w) regW			
ord regW = regZ, upper27_lower10		(r) regZ	(w) regW			
ord regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
ornd regW = regZ, extend27_upper27_lower10		(r) regZ	(w) regW			
ornd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
ornd regW = regZ, s10		(r) regZ	(w) regW			
ornd regW = regZ, upper27_lower10		(r) regZ	(w) regW			
ornd regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
ornw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
ornw regW = regZ, s10		(r) regZ	(w) regW			
ornw regW = regZ, upper27_lower10		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
orw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
orw regW = regZ, s10		(r) regZ	(w) regW			
orw regW = regZ, upper27_lower10		(r) regZ	(w) regW			
pcrel regW = extend27_upper27_lower10			(w) regW			
pcrel regW = extend6_upper27_lower10			(w) regW			
pcrel regW = s16			(w) regW			
rolw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
rolw regW = regZ, u6		(r) regZ	(w) regW			
rolwps regW = regZ, regY		(r) regZ (r) regY	(w) regW			
rolwps regW = regZ, u6		(r) regZ	(w) regW			
rorw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
rorw regW = regZ, u6		(r) regZ	(w) regW			
rorwps regW = regZ, regY		(r) regZ (r) regY	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
rorwps regW = regZ, u6		(r) regZ	(w) regW			
satd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
satd regW = regZ, u6		(r) regZ	(w) regW			
satdh regW = regZ		(r) regZ	(w) regW			
satdw regW = regZ		(r) regZ	(w) regW			
sbfcd regW = regZ, regY		(r) CS (r) regZ (r) regY	(w) CS (w) regW			
sbfcd regW = regZ, upper27_lower5		(r) CS (r) regZ	(w) CS (w) regW			
sbfcd.*i* regW = regZ, regY		(r) regZ (r) regY	(w) CS (w) regW	(r) CS		
sbfcd.*i* regW = regZ, upper27_lower5		(r) regZ	(w) CS (w) regW	(r) CS		
sbfd regW = regZ, extend27_upper27_lower10		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
sbfd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfd regW = regZ, s10		(r) regZ	(w) regW			
sbfd regW = regZ, upper27_lower10		(r) regZ	(w) regW			
sbfd regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
sbfhcp.*c* regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfhcp.*c* regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
sbfhq regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfhq regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
sbfsd regW = regZ, extend27_upper27_lower10		(r) regZ	(w) regW			
sbfsd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfsd regW = regZ, s10		(r) regZ	(w) regW			
sbfsd regW = regZ, upper27_lower10		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
sbfshq regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfshq regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
sbfsw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfsw regW = regZ, upper27_lower5		(r) regZ	(w) regW			
sbfswp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfswp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
sbfuwd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfuwd regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
sbfw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfw regW = regZ, s10		(r) regZ	(w) regW			
sbfw regW = regZ, upper27_lower10		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
sbfwc.*c* regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfwc.*c* regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
sbfwd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfwd regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
sbfwp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfwp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
sbfx16d regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfx16d regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
sbfx16hq regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfx16hq regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
sbfx16uwd regW = regZ, regY		(r) regZ (r) regY	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
sbfx16uwd regW = regZ, upper27_lower5		(r) regZ	(w) regW			
sbfx16w regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfx16w regW = regZ, upper27_lower5		(r) regZ	(w) regW			
sbfx16wd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfx16wd regW = regZ, upper27_lower5		(r) regZ	(w) regW			
sbfx16wp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfx16wp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
sbfx2d regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfx2d regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
sbfx2hq regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfx2hq regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
sbfx2uwd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfx2uwd regW = regZ, upper27_lower5		(r) regZ	(w) regW			
sbfx2w regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfx2w regW = regZ, upper27_lower5		(r) regZ	(w) regW			
sbfx2wd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfx2wd regW = regZ, upper27_lower5		(r) regZ	(w) regW			
sbfx2wp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfx2wp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
sbfx4d regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfx4d regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
sbfx4hq regW = regZ, regY		(r) regZ (r) regY	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
sbfx4hq regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
sbfx4uwd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfx4uwd regW = regZ, upper27_lower5		(r) regZ	(w) regW			
sbfx4w regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfx4w regW = regZ, upper27_lower5		(r) regZ	(w) regW			
sbfx4wd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfx4wd regW = regZ, upper27_lower5		(r) regZ	(w) regW			
sbfx4wp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfx4wp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
sbfx8d regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfx8d regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
sbfx8hq regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfx8hq regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
sbfx8uwd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfx8uwd regW = regZ, upper27_lower5		(r) regZ	(w) regW			
sbfx8w regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfx8w regW = regZ, upper27_lower5		(r) regZ	(w) regW			
sbfx8wd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfx8wd regW = regZ, upper27_lower5		(r) regZ	(w) regW			
sbfx8wp regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbfx8wp regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
sbmm8 regW = regZ, extend27_upper27_lower10		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
sbmm8 regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbmm8 regW = regZ, s10		(r) regZ	(w) regW			
sbmm8 regW = regZ, upper27_lower10		(r) regZ	(w) regW			
sbmm8 regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
sbmm8d regW = regZ, extend27_upper27_lower10		(r) regZ	(w) regW			
sbmm8d regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbmm8d regW = regZ, s10		(r) regZ	(w) regW			
sbmm8d regW = regZ, upper27_lower10		(r) regZ	(w) regW			
sbmm8d regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
sbmmt8 regW = regZ, extend27_upper27_lower10		(r) regZ	(w) regW			
sbmmt8 regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbmmt8 regW = regZ, s10		(r) regZ	(w) regW			
sbmmt8 regW = regZ, upper27_lower10		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
sbmmt8 regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
sbmmt8d regW = regZ, extend27_upper27_lower10		(r) regZ	(w) regW			
sbmmt8d regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sbmmt8d regW = regZ, s10		(r) regZ	(w) regW			
sbmmt8d regW = regZ, upper27_lower10		(r) regZ	(w) regW			
sbmmt8d regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
slld regW = regZ, regY		(r) regZ (r) regY	(w) regW			
slld regW = regZ, u6		(r) regZ	(w) regW			
sllhqs regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sllhqs regW = regZ, u6		(r) regZ	(w) regW			
sllw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sllw regW = regZ, u6		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
sllwps regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sllwps regW = regZ, u6		(r) regZ	(w) regW			
slsd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
slsd regW = regZ, u6		(r) regZ	(w) regW			
slshqs regW = regZ, regY		(r) regZ (r) regY	(w) regW			
slshqs regW = regZ, u6		(r) regZ	(w) regW			
slsw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
slsw regW = regZ, u6		(r) regZ	(w) regW			
slswps regW = regZ, regY		(r) regZ (r) regY	(w) regW			
slswps regW = regZ, u6		(r) regZ	(w) regW			
srad regW = regZ, regY		(r) regZ (r) regY	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
srad regW = regZ, u6		(r) regZ	(w) regW			
srahqs regW = regZ, regY		(r) regZ (r) regY	(w) regW			
srahqs regW = regZ, u6		(r) regZ	(w) regW			
sraw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sraw regW = regZ, u6		(r) regZ	(w) regW			
srawps regW = regZ, regY		(r) regZ (r) regY	(w) regW			
srawps regW = regZ, u6		(r) regZ	(w) regW			
srld regW = regZ, regY		(r) regZ (r) regY	(w) regW			
srld regW = regZ, u6		(r) regZ	(w) regW			
srlhqs regW = regZ, regY		(r) regZ (r) regY	(w) regW			
srlhqs regW = regZ, u6		(r) regZ	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
srlw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
srlw regW = regZ, u6		(r) regZ	(w) regW			
srlwps regW = regZ, regY		(r) regZ (r) regY	(w) regW			
srlwps regW = regZ, u6		(r) regZ	(w) regW			
srsd regW = regZ, regY		(r) regZ (r) regY	(w) regW			
srsd regW = regZ, u6		(r) regZ	(w) regW			
srshqs regW = regZ, regY		(r) regZ (r) regY	(w) regW			
srshqs regW = regZ, u6		(r) regZ	(w) regW			
srsw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
srsw regW = regZ, u6		(r) regZ	(w) regW			
srswps regW = regZ, regY		(r) regZ (r) regY	(w) regW			

Instruction	ID	RR	E1	E2	E3	E4
srswps regW = regZ, u6		(r) regZ	(w) regW			
stsud regW = regZ, regY		(r) regZ (r) regY	(w) regW			
stsuw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
sxbd regW = regZ		(r) regZ	(w) regW			
sxhd regW = regZ		(r) regZ	(w) regW			
sxlbhq regW = regZ		(r) regZ	(w) regW			
sxlhwp regW = regZ		(r) regZ	(w) regW			
sxmbhq regW = regZ		(r) regZ	(w) regW			
sxmhwp regW = regZ		(r) regZ	(w) regW			
sxwd regW = regZ		(r) regZ	(w) regW			
xmovetq regAE = regZ, regY		(r) regZ (r) regY	(w) CS (w) regAE			

Instruction	ID	RR	E1	E2	E3	E4
xmovetq regAO = regZ, regY		(r) regZ (r) regY	(w) CS (w) regAO			
xord regW = regZ, extend27_upper27_lower10		(r) regZ	(w) regW			
xord regW = regZ, regY		(r) regZ (r) regY	(w) regW			
xord regW = regZ, s10		(r) regZ	(w) regW			
xord regW = regZ, upper27_lower10		(r) regZ	(w) regW			
xord regW = regZ, upper27_lower5splat32		(r) regZ	(w) regW			
xorw regW = regZ, regY		(r) regZ (r) regY	(w) regW			
xorw regW = regZ, s10		(r) regZ	(w) regW			
xorw regW = regZ, upper27_lower10		(r) regZ	(w) regW			
zxbd regW = regZ		(r) regZ	(w) regW			
zxhd regW = regZ		(r) regZ	(w) regW			
zxwd regW = regZ		(r) regZ	(w) regW			

8.2 MAU instructions

Instruction	ID	RR	E1	E2	E3	E4
cmuldt regM = regZ, extend27_upper27_lower10		(r) regZ		(wd) regM		
cmuldt regM = regZ, regY		(r) regZ (r) regY		(wd) regM		
cmuldt regM = regZ, s10		(r) regZ		(wd) regM		
cmuldt regM = regZ, upper27_lower10		(r) regZ		(wd) regM		
cmulghxdt regM = regZ, regY		(r) regZ (r) regY	(rd) regM	(wd) regM		
cmulglxdt regM = regZ, regY		(r) regZ (r) regY	(rd) regM	(wd) regM		
cmulgmxdt regM = regZ, regY		(r) regZ (r) regY	(rd) regM	(wd) regM		
cmulxdt regM = regZ, regY		(r) regZ (r) regY	(rd) regM	(wd) regM		
copyq regM = regZ, regY		(r) regZ (r) regY		(wd) regM		
crcbellw regW = regZ, regY		(r) regZ (r) regY	(r) regW	(w) regW		

Instruction	ID	RR	E1	E2	E3	E4
crcbellw regW = regZ, upper27_lower5		(r) regZ	(r) regW	(w) regW		
crcbelmw regW = regZ, regY		(r) regZ (r) regY	(r) regW	(w) regW		
crcbelmw regW = regZ, upper27_lower5		(r) regZ	(r) regW	(w) regW		
crclellw regW = regZ, regY		(r) regZ (r) regY	(r) regW	(w) regW		
crclellw regW = regZ, upper27_lower5		(r) regZ	(r) regW	(w) regW		
crclelmw regW = regZ, regY		(r) regZ (r) regY	(r) regW	(w) regW		
crclelmw regW = regZ, upper27_lower5		(r) regZ	(r) regW	(w) regW		
dot2suwd regW = regZ, extend27_upper27_lower10		(r) regZ		(w) regW		
dot2suwd regW = regZ, regY		(r) regZ (r) regY		(w) regW		
dot2suwd regW = regZ, s10		(r) regZ		(w) regW		
dot2suwd regW = regZ, upper27_lower10		(r) regZ		(w) regW		
dot2suwdp regM = regP, regO		(rd) regP (rd) regO		(wd) regM		

Instruction	ID	RR	E1	E2	E3	E4
dot2uwd regW = regZ, extend27_upper27_lower10		(r) regZ		(w) regW		
dot2uwd regW = regZ, regY		(r) regZ (r) regY		(w) regW		
dot2uwd regW = regZ, s10		(r) regZ		(w) regW		
dot2uwd regW = regZ, upper27_lower10		(r) regZ		(w) regW		
dot2uwdp regM = regP, regO		(rd) regP (rd) regO		(wd) regM		
dot2w regW = regZ, extend27_upper27_lower10		(r) regZ		(w) regW		
dot2w regW = regZ, regY		(r) regZ (r) regY		(w) regW		
dot2w regW = regZ, s10		(r) regZ		(w) regW		
dot2w regW = regZ, upper27_lower10		(r) regZ		(w) regW		
dot2wd regW = regZ, extend27_upper27_lower10		(r) regZ		(w) regW		
dot2wd regW = regZ, regY		(r) regZ (r) regY		(w) regW		
dot2wd regW = regZ, s10		(r) regZ		(w) regW		

Instruction	ID	RR	E1	E2	E3	E4
dot2wd regW = regZ, upper27_lower10		(r) regZ		(w) regW		
dot2wdp regM = regP, regO		(rd) regP (rd) regO		(wd) regM		
dot2wzp regM = regP, regO		(rd) regP (rd) regO		(wd) regM		
faddd regW = regZ, extend27_upper27_lower10		(r) regZ				(w) regW
faddd regW = regZ, s10		(r) regZ				(w) regW
faddd regW = regZ, upper27_lower10		(r) regZ				(w) regW
fadddroundingsilent regW = regZ, regY		(r) regZ (r) regY				(w) regW
fadddcroundingsilent regM = regP, regO		(rd) regP (rd) regO				(wd) regM
fadddc.*croundingsilent* regM = regP, regO		(rd) regP (rd) regO				(wd) regM
fadddproundingsilent regM = regP, regO		(rd) regP (rd) regO				(wd) regM
faddhq regW = regZ, extend27_upper27_lower10		(r) regZ				(w) regW

Instruction	ID	RR	E1	E2	E3	E4
faddhq regW = regZ, s10		(r) regZ				(w) regW
faddhq regW = regZ, upper27_lower10		(r) regZ				(w) regW
faddhqroundingsilent regW = regZ, regY		(r) regZ (r) regY				(w) regW
faddw regW = regZ, extend27_upper27_lower10		(r) regZ				(w) regW
faddw regW = regZ, s10		(r) regZ				(w) regW
faddw regW = regZ, upper27_lower10		(r) regZ				(w) regW
faddwroundingsilent regW = regZ, regY		(r) regZ (r) regY				(w) regW
faddwcroundingsilent regW = regZ, regY		(r) regZ (r) regY				(w) regW
faddwc.*c* regW = regZ, extend27_upper27_lower10		(r) regZ				(w) regW
faddwc.*c* regW = regZ, s10		(r) regZ				(w) regW
faddwc.*c* regW = regZ, upper27_lower10		(r) regZ				(w) regW
faddwc.*croundingsilent* regW = regZ, regY		(r) regZ (r) regY				(w) regW

Instruction	ID	RR	E1	E2	E3	E4
faddwcproundingsilent regM = regP, regO		(rd) regP (rd) regO				(wd) regM
faddwcp.*croundingsilent* regM = regP, regO		(rd) regP (rd) regO				(wd) regM
faddwp regW = regZ, extend27_upper27_lower10		(r) regZ				(w) regW
faddwp regW = regZ, s10		(r) regZ				(w) regW
faddwp regW = regZ, upper27_lower10		(r) regZ				(w) regW
faddwproundingsilent regW = regZ, regY		(r) regZ (r) regY				(w) regW
faddwqroundingsilent regM = regP, regO		(rd) regP (rd) regO				(wd) regM
fdot2w regW = regZ, extend27_upper27_lower10		(r) regZ				(w) regW
fdot2w regW = regZ, s10		(r) regZ				(w) regW
fdot2w regW = regZ, upper27_lower10		(r) regZ				(w) regW
fdot2wroundingsilent regW = regZ, regY		(r) regZ (r) regY				(w) regW
fdot2wd regW = regZ, extend27_upper27_lower10		(r) regZ				(w) regW

Instruction	ID	RR	E1	E2	E3	E4
fdot2wd regW = regZ, s10		(r) regZ				(w) regW
fdot2wd regW = regZ, upper27_lower10		(r) regZ				(w) regW
fdot2wdroundingsilent regW = regZ, regY		(r) regZ (r) regY				(w) regW
fdot2wdproundingsilent regM = regP, regO		(rd) regP (rd) regO				(wd) regM
fdot2wzproundingsilent regM = regP, regO		(rd) regP (rd) regO				(wd) regM
ffmad regW = regZ, extend27_upper27_lower10		(r) regW (r) regZ				(w) regW
ffmad regW = regZ, s10		(r) regW (r) regZ				(w) regW
ffmad regW = regZ, upper27_lower10		(r) regW (r) regZ				(w) regW
ffmadroundingsilent regW = regZ, regY		(r) regW (r) regZ (r) regY				(w) regW
ffmahq regW = regZ, extend27_upper27_lower10		(r) regW (r) regZ				(w) regW

Instruction	ID	RR	E1	E2	E3	E4
ffmahq regW = regZ, s10		(r) regW (r) regZ				(w) regW
ffmahq regW = regZ, upper27_lower10		(r) regW (r) regZ				(w) regW
ffmahqroundingsilent regW = regZ, regY		(r) regW (r) regZ (r) regY				(w) regW
ffmahw regW = regZ, extend27_upper27_lower10		(r) regW (r) regZ				(w) regW
ffmahw regW = regZ, s10		(r) regW (r) regZ				(w) regW
ffmahw regW = regZ, upper27_lower10		(r) regW (r) regZ				(w) regW
ffmahwroundingsilent regW = regZ, regY		(r) regW (r) regZ (r) regY				(w) regW
ffmahwq regM = regZ, extend27_upper27_lower10		(rd) regM (r) regZ				(wd) regM
ffmahwq regM = regZ, s10		(rd) regM (r) regZ				(wd) regM

Instruction	ID	RR	E1	E2	E3	E4
ffmahwq regM = regZ, upper27_lower10		(rd) regM (r) regZ				(wd) regM
ffmahwqroundingsilent regM = regZ, regY		(rd) regM (r) regZ (r) regY				(wd) regM
ffmaw regW = regZ, extend27_upper27_lower10		(r) regW (r) regZ				(w) regW
ffmaw regW = regZ, s10		(r) regW (r) regZ				(w) regW
ffmaw regW = regZ, upper27_lower10		(r) regW (r) regZ				(w) regW
ffmawroundingsilent regW = regZ, regY		(r) regW (r) regZ (r) regY				(w) regW
ffmawd regW = regZ, extend27_upper27_lower10		(r) regW (r) regZ				(w) regW
ffmawd regW = regZ, s10		(r) regW (r) regZ				(w) regW
ffmawd regW = regZ, upper27_lower10		(r) regW (r) regZ				(w) regW

Instruction	ID	RR	E1	E2	E3	E4
ffmawdroundingsilent regW = regZ, regY		(r) regW (r) regZ (r) regY				(w) regW
ffmawdp regM = regZ, extend27_upper27_lower10		(rd) regM (r) regZ				(wd) regM
ffmawdp regM = regZ, s10		(rd) regM (r) regZ				(wd) regM
ffmawdp regM = regZ, upper27_lower10		(rd) regM (r) regZ				(wd) regM
ffmawdproundingsilent regM = regZ, regY		(rd) regM (r) regZ (r) regY				(wd) regM
ffmawp regW = regZ, extend27_upper27_lower10		(r) regW (r) regZ				(w) regW
ffmawp regW = regZ, s10		(r) regW (r) regZ				(w) regW
ffmawp regW = regZ, upper27_lower10		(r) regW (r) regZ				(w) regW

Instruction	ID	RR	E1	E2	E3	E4
ffmawproundingsilent regW = regZ, regY		(r) regW (r) regZ (r) regY				(w) regW
ffmsd regW = regZ, extend27_upper27_lower10		(r) regW (r) regZ				(w) regW
ffmsd regW = regZ, s10		(r) regW (r) regZ				(w) regW
ffmsd regW = regZ, upper27_lower10		(r) regW (r) regZ				(w) regW
ffmsdroundingsilent regW = regZ, regY		(r) regW (r) regZ (r) regY				(w) regW
ffmshq regW = regZ, extend27_upper27_lower10		(r) regW (r) regZ				(w) regW
ffmshq regW = regZ, s10		(r) regW (r) regZ				(w) regW
ffmshq regW = regZ, upper27_lower10		(r) regW (r) regZ				(w) regW

Instruction	ID	RR	E1	E2	E3	E4
ffmshqroundingsilent regW = regZ, regY		(r) regW (r) regZ (r) regY				(w) regW
ffmshw regW = regZ, extend27_upper27_lower10		(r) regW (r) regZ				(w) regW
ffmshw regW = regZ, s10		(r) regW (r) regZ				(w) regW
ffmshw regW = regZ, upper27_lower10		(r) regW (r) regZ				(w) regW
ffmshwroundingsilent regW = regZ, regY		(r) regW (r) regZ (r) regY				(w) regW
ffmshwq regM = regZ, extend27_upper27_lower10		(rd) regM (r) regZ				(wd) regM
ffmshwq regM = regZ, s10		(rd) regM (r) regZ				(wd) regM
ffmshwq regM = regZ, upper27_lower10		(rd) regM (r) regZ				(wd) regM

Instruction	ID	RR	E1	E2	E3	E4
ffmshwqroundingsilent regM = regZ, regY		(rd) regM (r) regZ (r) regY				(wd) regM
ffmsw regW = regZ, extend27_upper27_lower10		(r) regW (r) regZ				(w) regW
ffmsw regW = regZ, s10		(r) regW (r) regZ				(w) regW
ffmsw regW = regZ, upper27_lower10		(r) regW (r) regZ				(w) regW
ffmswroundingsilent regW = regZ, regY		(r) regW (r) regZ (r) regY				(w) regW
ffmswd regW = regZ, extend27_upper27_lower10		(r) regW (r) regZ				(w) regW
ffmswd regW = regZ, s10		(r) regW (r) regZ				(w) regW
ffmswd regW = regZ, upper27_lower10		(r) regW (r) regZ				(w) regW

Instruction	ID	RR	E1	E2	E3	E4
ffmswdroundingsilent regW = regZ, regY		(r) regW (r) regZ (r) regY				(w) regW
ffmswdp regM = regZ, extend27_upper27_lower10		(rd) regM (r) regZ				(wd) regM
ffmswdp regM = regZ, s10		(rd) regM (r) regZ				(wd) regM
ffmswdp regM = regZ, upper27_lower10		(rd) regM (r) regZ				(wd) regM
ffmswdproundingsilent regM = regZ, regY		(rd) regM (r) regZ (r) regY				(wd) regM
ffmswp regW = regZ, extend27_upper27_lower10		(r) regW (r) regZ				(w) regW
ffmswp regW = regZ, s10		(r) regW (r) regZ				(w) regW
ffmswp regW = regZ, upper27_lower10		(r) regW (r) regZ				(w) regW

Instruction	ID	RR	E1	E2	E3	E4
ffmswproundingsilent regW = regZ, regY		(r) regW (r) regZ (r) regY				(w) regW
fixeddroundingsilent regW = regZ, u6		(r) regZ				(w) regW
fixedudroundingsilent regW = regZ, u6		(r) regZ				(w) regW
fixeduwroundingsilent regW = regZ, u6		(r) regZ				(w) regW
fixeduwproundingsilent regW = regZ, u6		(r) regZ				(w) regW
fixedwroundingsilent regW = regZ, u6		(r) regZ				(w) regW
fixedwproundingsilent regW = regZ, u6		(r) regZ				(w) regW
floatdroundingsilent regW = regZ, u6		(r) regZ				(w) regW
floatudroundingsilent regW = regZ, u6		(r) regZ				(w) regW
floatuwroundingsilent regW = regZ, u6		(r) regZ				(w) regW
floatuwproundingsilent regW = regZ, u6		(r) regZ				(w) regW
floatwroundingsilent regW = regZ, u6		(r) regZ				(w) regW
floatwproundingsilent regW = regZ, u6		(r) regZ				(w) regW

Instruction	ID	RR	E1	E2	E3	E4
fmm212wroundingsilent regM = regZ, regY		(r) regZ (r) regY				(wd) regM
fmma212wroundingsilent regM = regZ, regY		(rd) regM (r) regZ (r) regY				(wd) regM
fmms212wroundingsilent regM = regZ, regY		(rd) regM (r) regZ (r) regY				(wd) regM
fmuld regW = regZ, extend27_upper27_lower10		(r) regZ				(w) regW
fmuld regW = regZ, s10		(r) regZ				(w) regW
fmuld regW = regZ, upper27_lower10		(r) regZ				(w) regW
fmuldroundingsilent regW = regZ, regY		(r) regZ (r) regY				(w) regW
fmulhq regW = regZ, extend27_upper27_lower10		(r) regZ				(w) regW
fmulhq regW = regZ, s10		(r) regZ				(w) regW
fmulhq regW = regZ, upper27_lower10		(r) regZ				(w) regW
fmulhqroundingsilent regW = regZ, regY		(r) regZ (r) regY				(w) regW

Instruction	ID	RR	E1	E2	E3	E4
fmulhw regW = regZ, extend27_upper27_lower10		(r) regZ				(w) regW
fmulhw regW = regZ, s10		(r) regZ				(w) regW
fmulhw regW = regZ, upper27_lower10		(r) regZ				(w) regW
fmulhwroundingsilent regW = regZ, regY		(r) regZ (r) regY				(w) regW
fmulhwq regM = regZ, extend27_upper27_lower10		(r) regZ				(wd) regM
fmulhwq regM = regZ, s10		(r) regZ				(wd) regM
fmulhwq regM = regZ, upper27_lower10		(r) regZ				(wd) regM
fmulhwqroundingsilent regM = regZ, regY		(r) regZ (r) regY				(wd) regM
fmulw regW = regZ, extend27_upper27_lower10		(r) regZ				(w) regW
fmulw regW = regZ, s10		(r) regZ				(w) regW
fmulw regW = regZ, upper27_lower10		(r) regZ				(w) regW
fmulwroundingsilent regW = regZ, regY		(r) regZ (r) regY				(w) regW
fmulwc regW = regZ, extend27_upper27_lower10		(r) regZ				(w) regW

Instruction	ID	RR	E1	E2	E3	E4
fmulwc regW = regZ, s10		(r) regZ				(w) regW
fmulwc regW = regZ, upper27_lower10		(r) regZ				(w) regW
fmulwcroundingsilent regW = regZ, regY		(r) regZ (r) regY				(w) regW
fmulwc.*c* regW = regZ, extend27_upper27_lower10		(r) regZ				(w) regW
fmulwc.*c* regW = regZ, s10		(r) regZ				(w) regW
fmulwc.*c* regW = regZ, upper27_lower10		(r) regZ				(w) regW
fmulwc.*croundingsilent* regW = regZ, regY		(r) regZ (r) regY				(w) regW
fmulwd regW = regZ, extend27_upper27_lower10		(r) regZ				(w) regW
fmulwd regW = regZ, s10		(r) regZ				(w) regW
fmulwd regW = regZ, upper27_lower10		(r) regZ				(w) regW
fmulwdroundingsilent regW = regZ, regY		(r) regZ (r) regY				(w) regW
fmulwdc regM = regZ, extend27_upper27_lower10		(r) regZ				(wd) regM
fmulwdc regM = regZ, s10		(r) regZ				(wd) regM

Instruction	ID	RR	E1	E2	E3	E4
fmulwdc regM = regZ, upper27_lower10		(r) regZ				(wd) regM
fmulwdcroundingsilent regM = regZ, regY		(r) regZ (r) regY				(wd) regM
fmulwdc.*c* regM = regZ, extend27_upper27_lower10		(r) regZ				(wd) regM
fmulwdc.*c* regM = regZ, s10		(r) regZ				(wd) regM
fmulwdc.*c* regM = regZ, upper27_lower10		(r) regZ				(wd) regM
fmulwdc.*croundingsilent* regM = regZ, regY		(r) regZ (r) regY				(wd) regM
fmulwdp regM = regZ, extend27_upper27_lower10		(r) regZ				(wd) regM
fmulwdp regM = regZ, s10		(r) regZ				(wd) regM
fmulwdp regM = regZ, upper27_lower10		(r) regZ				(wd) regM
fmulwdproundingsilent regM = regZ, regY		(r) regZ (r) regY				(wd) regM
fmulwp regW = regZ, extend27_upper27_lower10		(r) regZ				(w) regW
fmulwp regW = regZ, s10		(r) regZ				(w) regW
fmulwp regW = regZ, upper27_lower10		(r) regZ				(w) regW

Instruction	ID	RR	E1	E2	E3	E4
fmulwproundingsilent regW = regZ, regY		(r) regZ (r) regY				(w) regW
fmulwqroundingsilent regM = regP, regO		(rd) regP (rd) regO				(wd) regM
fsbfd regW = regZ, extend27_upper27_lower10		(r) regZ				(w) regW
fsbfd regW = regZ, s10		(r) regZ				(w) regW
fsbfd regW = regZ, upper27_lower10		(r) regZ				(w) regW
fsbfdroundingsilent regW = regZ, regY		(r) regZ (r) regY				(w) regW
fsbfdcroundingsilent regM = regP, regO		(rd) regP (rd) regO				(wd) regM
fsbfdc.*croundingsilent* regM = regP, regO		(rd) regP (rd) regO				(wd) regM
fsbfdproundingsilent regM = regP, regO		(rd) regP (rd) regO				(wd) regM
fsbfhq regW = regZ, extend27_upper27_lower10		(r) regZ				(w) regW
fsbfhq regW = regZ, s10		(r) regZ				(w) regW

Instruction	ID	RR	E1	E2	E3	E4
fsbfhq regW = regZ, upper27_lower10		(r) regZ				(w) regW
fsbfhqroundingsilent regW = regZ, regY		(r) regZ (r) regY				(w) regW
fsbfw regW = regZ, extend27_upper27_lower10		(r) regZ				(w) regW
fsbfw regW = regZ, s10		(r) regZ				(w) regW
fsbfw regW = regZ, upper27_lower10		(r) regZ				(w) regW
fsbfwroundingsilent regW = regZ, regY		(r) regZ (r) regY				(w) regW
fsbfwcroundingsilent regW = regZ, regY		(r) regZ (r) regY				(w) regW
fsbfwc.*c* regW = regZ, extend27_upper27_lower10		(r) regZ				(w) regW
fsbfwc.*c* regW = regZ, s10		(r) regZ				(w) regW
fsbfwc.*c* regW = regZ, upper27_lower10		(r) regZ				(w) regW
fsbfwc.*croundingsilent* regW = regZ, regY		(r) regZ (r) regY				(w) regW
fsbfwcproundingsilent regM = regP, regO		(rd) regP (rd) regO				(wd) regM

Instruction	ID	RR	E1	E2	E3	E4
fsbfwcp.*croundingsilent* regM = regP, regO		(rd) regP (rd) regO				(wd) regM
fsbfwp regW = regZ, extend27_upper27_lower10		(r) regZ				(w) regW
fsbfwp regW = regZ, s10		(r) regZ				(w) regW
fsbfwp regW = regZ, upper27_lower10		(r) regZ				(w) regW
fsbfwproundingsilent regW = regZ, regY		(r) regZ (r) regY				(w) regW
fsbfwqroundingsilent regM = regP, regO		(rd) regP (rd) regO				(wd) regM
maddd regW = regZ, extend27_upper27_lower10		(r) regZ	(r) regW	(w) regW		
maddd regW = regZ, regY		(r) regZ (r) regY	(r) regW	(w) regW		
maddd regW = regZ, s10		(r) regZ	(r) regW	(w) regW		
maddd regW = regZ, upper27_lower10		(r) regZ	(r) regW	(w) regW		
madddt regM = regZ, extend27_upper27_lower10		(r) regZ	(rd) regM	(wd) regM		
madddt regM = regZ, regY		(r) regZ (r) regY	(rd) regM	(wd) regM		

Instruction	ID	RR	E1	E2	E3	E4
madddt regM = regZ, s10		(r) regZ	(rd) regM	(wd) regM		
madddt regM = regZ, upper27_lower10		(r) regZ	(rd) regM	(wd) regM		
maddhq regW = regZ, extend27_upper27_lower10		(r) regZ	(r) regW	(w) regW		
maddhq regW = regZ, regY		(r) regZ (r) regY	(r) regW	(w) regW		
maddhq regW = regZ, s10		(r) regZ	(r) regW	(w) regW		
maddhq regW = regZ, upper27_lower10		(r) regZ	(r) regW	(w) regW		
maddhwq regM = regZ, regY		(r) regZ (r) regY	(rd) regM	(wd) regM		
maddsudt regM = regZ, extend27_upper27_lower10		(r) regZ	(rd) regM	(wd) regM		
maddsudt regM = regZ, regY		(r) regZ (r) regY	(rd) regM	(wd) regM		
maddsudt regM = regZ, s10		(r) regZ	(rd) regM	(wd) regM		
maddsudt regM = regZ, upper27_lower10		(r) regZ	(rd) regM	(wd) regM		
maddsuhwq regM = regZ, regY		(r) regZ (r) regY	(rd) regM	(wd) regM		

Instruction	ID	RR	E1	E2	E3	E4
maddsuwd regW = regZ, regY		(r) regZ (r) regY	(r) regW	(w) regW		
maddsuwd regW = regZ, upper27_lower5		(r) regZ	(r) regW	(w) regW		
maddsuwdp regM = regZ, regY		(r) regZ (r) regY	(rd) regM	(wd) regM		
maddsuwdp regM = regZ, upper27_lower5splat32		(r) regZ	(rd) regM	(wd) regM		
maddudt regM = regZ, extend27_upper27_lower10		(r) regZ	(rd) regM	(wd) regM		
maddudt regM = regZ, regY		(r) regZ (r) regY	(rd) regM	(wd) regM		
maddudt regM = regZ, s10		(r) regZ	(rd) regM	(wd) regM		
maddudt regM = regZ, upper27_lower10		(r) regZ	(rd) regM	(wd) regM		
madduhwq regM = regZ, regY		(r) regZ (r) regY	(rd) regM	(wd) regM		
madduwd regW = regZ, regY		(r) regZ (r) regY	(r) regW	(w) regW		
madduwd regW = regZ, upper27_lower5		(r) regZ	(r) regW	(w) regW		

Instruction	ID	RR	E1	E2	E3	E4
madduwdp regM = regZ, regY		(r) regZ (r) regY	(rd) regM	(wd) regM		
madduwdp regM = regZ, upper27_lower5splat32		(r) regZ	(rd) regM	(wd) regM		
madduzdt regM = regZ, extend27_upper27_lower10		(r) regZ	(rd) regM	(wd) regM		
madduzdt regM = regZ, regY		(r) regZ (r) regY	(rd) regM	(wd) regM		
madduzdt regM = regZ, s10		(r) regZ	(rd) regM	(wd) regM		
madduzdt regM = regZ, upper27_lower10		(r) regZ	(rd) regM	(wd) regM		
maddw regW = regZ, regY		(r) regZ (r) regY	(r) regW	(w) regW		
maddw regW = regZ, upper27_lower5		(r) regZ	(r) regW	(w) regW		
maddwd regW = regZ, regY		(r) regZ (r) regY	(r) regW	(w) regW		
maddwd regW = regZ, upper27_lower5		(r) regZ	(r) regW	(w) regW		
maddwdp regM = regZ, regY		(r) regZ (r) regY	(rd) regM	(wd) regM		
maddwdp regM = regZ, upper27_lower5splat32		(r) regZ	(rd) regM	(wd) regM		

Instruction	ID	RR	E1	E2	E3	E4
maddwp regW = regZ, extend27_upper27_lower10		(r) regZ	(r) regW	(w) regW		
maddwp regW = regZ, regY		(r) regZ (r) regY	(r) regW	(w) regW		
maddwp regW = regZ, s10		(r) regZ	(r) regW	(w) regW		
maddwp regW = regZ, upper27_lower10		(r) regZ	(r) regW	(w) regW		
mm212w regM = regZ, regY		(r) regZ (r) regY		(wd) regM		
mm212w regM = regZ, upper27_lower5splat32		(r) regZ		(wd) regM		
mma212w regM = regZ, regY		(r) regZ (r) regY	(rd) regM	(wd) regM		
mma212w regM = regZ, upper27_lower5splat32		(r) regZ	(rd) regM	(wd) regM		
mms212w regM = regZ, regY		(r) regZ (r) regY	(rd) regM	(wd) regM		
mms212w regM = regZ, upper27_lower5splat32		(r) regZ	(rd) regM	(wd) regM		
msbfd regW = regZ, regY		(r) regZ (r) regY	(r) regW	(w) regW		
msbfd regW = regZ, upper27_lower5splat32		(r) regZ	(r) regW	(w) regW		

Instruction	ID	RR	E1	E2	E3	E4
msbfdt regM = regZ, regY		(r) regZ (r) regY	(rd) regM	(wd) regM		
msbfhq regW = regZ, regY		(r) regZ (r) regY	(r) regW	(w) regW		
msbfhq regW = regZ, upper27_lower5splat32		(r) regZ	(r) regW	(w) regW		
msbfhwq regM = regZ, regY		(r) regZ (r) regY	(rd) regM	(wd) regM		
msbfsudt regM = regZ, regY		(r) regZ (r) regY	(rd) regM	(wd) regM		
msbfsuhwq regM = regZ, regY		(r) regZ (r) regY	(rd) regM	(wd) regM		
msbfsuwd regW = regZ, regY		(r) regZ (r) regY	(r) regW	(w) regW		
msbfsuwd regW = regZ, upper27_lower5		(r) regZ	(r) regW	(w) regW		
msbfsuwdp regM = regZ, regY		(r) regZ (r) regY	(rd) regM	(wd) regM		
msbfsuwdp regM = regZ, upper27_lower5splat32		(r) regZ	(rd) regM	(wd) regM		

Instruction	ID	RR	E1	E2	E3	E4
msbfudt regM = regZ, regY		(r) regZ (r) regY	(rd) regM	(wd) regM		
msbfuhwq regM = regZ, regY		(r) regZ (r) regY	(rd) regM	(wd) regM		
msbfuwd regW = regZ, regY		(r) regZ (r) regY	(r) regW	(w) regW		
msbfuwd regW = regZ, upper27_lower5		(r) regZ	(r) regW	(w) regW		
msbfuwdp regM = regZ, regY		(r) regZ (r) regY	(rd) regM	(wd) regM		
msbfuwdp regM = regZ, upper27_lower5splat32		(r) regZ	(rd) regM	(wd) regM		
msbfuzdt regM = regZ, regY		(r) regZ (r) regY	(rd) regM	(wd) regM		
msbfw regW = regZ, regY		(r) regZ (r) regY	(r) regW	(w) regW		
msbfw regW = regZ, upper27_lower5		(r) regZ	(r) regW	(w) regW		
msbfwd regW = regZ, regY		(r) regZ (r) regY	(r) regW	(w) regW		
msbfwd regW = regZ, upper27_lower5		(r) regZ	(r) regW	(w) regW		

Instruction	ID	RR	E1	E2	E3	E4
msbfwdp regM = regZ, regY		(r) regZ (r) regY	(rd) regM	(wd) regM		
msbfwdp regM = regZ, upper27_lower5splat32		(r) regZ	(rd) regM	(wd) regM		
msbfwp regW = regZ, regY		(r) regZ (r) regY	(r) regW	(w) regW		
msbfwp regW = regZ, upper27_lower5splat32		(r) regZ	(r) regW	(w) regW		
muld regW = regZ, extend27_upper27_lower10		(r) regZ		(w) regW		
muld regW = regZ, regY		(r) regZ (r) regY		(w) regW		
muld regW = regZ, s10		(r) regZ		(w) regW		
muld regW = regZ, upper27_lower10		(r) regZ		(w) regW		
muldt regM = regZ, extend27_upper27_lower10		(r) regZ		(wd) regM		
muldt regM = regZ, regY		(r) regZ (r) regY		(wd) regM		
muldt regM = regZ, s10		(r) regZ		(wd) regM		
muldt regM = regZ, upper27_lower10		(r) regZ		(wd) regM		

Instruction	ID	RR	E1	E2	E3	E4
mulhq regW = regZ, extend27_upper27_lower10		(r) regZ		(w) regW		
mulhq regW = regZ, regY		(r) regZ (r) regY		(w) regW		
mulhq regW = regZ, s10		(r) regZ		(w) regW		
mulhq regW = regZ, upper27_lower10		(r) regZ		(w) regW		
mulhwq regM = regZ, regY		(r) regZ (r) regY		(wd) regM		
mulsudt regM = regZ, extend27_upper27_lower10		(r) regZ		(wd) regM		
mulsudt regM = regZ, regY		(r) regZ (r) regY		(wd) regM		
mulsudt regM = regZ, s10		(r) regZ		(wd) regM		
mulsudt regM = regZ, upper27_lower10		(r) regZ		(wd) regM		
mulsuhwq regM = regZ, regY		(r) regZ (r) regY		(wd) regM		
mulsuwd regW = regZ, regY		(r) regZ (r) regY		(w) regW		
mulsuwd regW = regZ, upper27_lower5		(r) regZ		(w) regW		

Instruction	ID	RR	E1	E2	E3	E4
mulsuwdp regM = regZ, regY		(r) regZ (r) regY		(wd) regM		
mulsuwdp regM = regZ, upper27_lower5splat32		(r) regZ		(wd) regM		
muludt regM = regZ, extend27_upper27_lower10		(r) regZ		(wd) regM		
muludt regM = regZ, regY		(r) regZ (r) regY		(wd) regM		
muludt regM = regZ, s10		(r) regZ		(wd) regM		
muludt regM = regZ, upper27_lower10		(r) regZ		(wd) regM		
muluhwq regM = regZ, regY		(r) regZ (r) regY		(wd) regM		
muluwd regW = regZ, regY		(r) regZ (r) regY		(w) regW		
muluwd regW = regZ, upper27_lower5		(r) regZ		(w) regW		
muluwdp regM = regZ, regY		(r) regZ (r) regY		(wd) regM		
muluwdp regM = regZ, upper27_lower5splat32		(r) regZ		(wd) regM		

Instruction	ID	RR	E1	E2	E3	E4
mulw regW = regZ, regY		(r) regZ (r) regY		(w) regW		
mulw regW = regZ, upper27_lower5		(r) regZ		(w) regW		
mulwc regW = regZ, extend27_upper27_lower10		(r) regZ		(w) regW		
mulwc regW = regZ, regY		(r) regZ (r) regY		(w) regW		
mulwc regW = regZ, s10		(r) regZ		(w) regW		
mulwc regW = regZ, upper27_lower10		(r) regZ		(w) regW		
mulwc.*c* regW = regZ, regY		(r) regZ (r) regY		(w) regW		
mulwd regW = regZ, regY		(r) regZ (r) regY		(w) regW		
mulwd regW = regZ, upper27_lower5		(r) regZ		(w) regW		
mulwdc regM = regZ, regY		(r) regZ (r) regY		(wd) regM		
mulwdc.*c* regM = regZ, regY		(r) regZ (r) regY		(wd) regM		

Instruction	ID	RR	E1	E2	E3	E4
mulwdp regM = regZ, regY		(r) regZ (r) regY		(wd) regM		
mulwdp regM = regZ, upper27_lower5splat32		(r) regZ		(wd) regM		
mulwp regW = regZ, extend27_upper27_lower10		(r) regZ		(w) regW		
mulwp regW = regZ, regY		(r) regZ (r) regY		(w) regW		
mulwp regW = regZ, s10		(r) regZ		(w) regW		
mulwp regW = regZ, upper27_lower10		(r) regZ		(w) regW		
mulwq regM = regP, regO		(rd) regP (rd) regO		(wd) regM		

8.3 LSU instructions

In the table below, an ”E11” column has been added to show when the results of uncached loads are available for the typical case if streaming is activated. If streaming is not activated, then uncached loads are blocking and will stall at E3 (and the upstream stages of the pipeline with them) for 7 cycles (typical), see section 4.1.2 for more details.

Instruction	ID	RR	E1	E2	E3	E4	...	E11
acswapddoscale regY[regZ] = regU		(r) regY (r) regZ	(rd) regU					
acswapd extend27_upper27_lower10[regZ] = regU		(r) regZ	(rd) regU					
acswapdlsucond regY? extend27_offset27[regZ] = regU		(r) regY (r) regZ	(rd) regU					
acswapdlsucond regY? offset27[regZ] = regU		(r) regY (r) regZ	(rd) regU					
acswapdlsucond regY? [regZ] = regU		(r) regY (r) regZ	(rd) regU					
acswapd s10[regZ] = regU		(r) regZ	(rd) regU					
acswapd upper27_lower10[regZ] = regU		(r) regZ	(rd) regU					
acswapwdoscale regY[regZ] = regU		(r) regY (r) regZ	(rd) regU					
acswapw extend27_upper27_lower10[regZ] = regU		(r) regZ	(rd) regU					

Instruction	ID	RR	E1	E2	E3	E4	...	E11
acswapwlsucond regY? extend27_offset27[regZ] = regU		(r) regY (r) regZ	(rd) regU					
acswapwlsucond regY? offset27[regZ] = regU		(r) regY (r) regZ	(rd) regU					
acswapwlsucond regY? [regZ] = regU		(r) regY (r) regZ	(rd) regU					
acswapw s10[regZ] = regU		(r) regZ	(rd) regU					
acswapw upper27_lower10[regZ] = regU		(r) regZ	(rd) regU					
aladdddoscale regY[regZ] = regT		(r) regY (r) regZ	(r) regT					
aladdd extend27_upper27_lower10[regZ] = regT		(r) regZ	(r) regT					
aladddlsucond regY? extend27_offset27[regZ] = regT		(r) regY (r) regZ	(r) regT					
aladddlsucond regY? offset27[regZ] = regT		(r) regY (r) regZ	(r) regT					
aladddlsucond regY? [regZ] = regT		(r) regY (r) regZ	(r) regT					
aladdd s10[regZ] = regT		(r) regZ	(r) regT					

Instruction	ID	RR	E1	E2	E3	E4	...	E11
aladdd upper27_lower10[regZ] = regT		(r) regZ	(r) regT					
aladdwdoscale regY[regZ] = regT		(r) regY (r) regZ	(r) regT					
aladdw extend27_upper27_lower10[regZ] = regT		(r) regZ	(r) regT					
aladdwlsucond regY? extend27_offset27[regZ] = regT		(r) regY (r) regZ	(r) regT					
aladdwlsucond regY? offset27[regZ] = regT		(r) regY (r) regZ	(r) regT					
aladdwlsucond regY? [regZ] = regT		(r) regY (r) regZ	(r) regT					
aladdw s10[regZ] = regT		(r) regZ	(r) regT					
aladdw upper27_lower10[regZ] = regT		(r) regZ	(r) regT					
alclrddoscale regW = regY[regZ]		(r) regY (r) regZ			(w) regW			
alclrdlsucond regY? regW = extend27_offset27[regZ]		(r) regY (r) regZ			(w) regW			
alclrdlsucond regY? regW = offset27[regZ]		(r) regY (r) regZ			(w) regW			

Instruction	ID	RR	E1	E2	E3	E4	...	E11
alclrdlsucond regY? regW = [regZ]		(r) regY (r) regZ			(w) regW			
alclrd regW = extend27_upper27_lower10[regZ]		(r) regZ			(w) regW			
alclrd regW = s10[regZ]		(r) regZ			(w) regW			
alclrd regW = upper27_lower10[regZ]		(r) regZ			(w) regW			
alclrwdoscale regW = regY[regZ]		(r) regY (r) regZ			(w) regW			
alclrwlsucond regY? regW = extend27_offset27[regZ]		(r) regY (r) regZ			(w) regW			
alclrwlsucond regY? regW = offset27[regZ]		(r) regY (r) regZ			(w) regW			
alclrwlsucond regY? regW = [regZ]		(r) regY (r) regZ			(w) regW			
alclrw regW = extend27_upper27_lower10[regZ]		(r) regZ			(w) regW			
alclrw regW = s10[regZ]		(r) regZ			(w) regW			
alclrw regW = upper27_lower10[regZ]		(r) regZ			(w) regW			
copyo regN = regR		(r) regR			(w) regN			

Instruction	ID	RR	E1	E2	E3	E4	...	E11
d1inval								
dinvall extend27_upper27_lower10[regZ]	(r) regZ							
dinvalllsucond regY? extend27_offset27[regZ]		(r) regY (r) regZ						
dinvalllsucond regY? offset27[regZ]		(r) regY (r) regZ						
dinvalllsucond regY? [regZ]		(r) regY (r) regZ						
dinvall regY[regZ]	(r) regY	(r) regZ						
dinvall s10[regZ]	(r) regZ							
dinvall upper27_lower10[regZ]	(r) regZ							
dtouchl extend27_upper27_lower10[regZ]	(r) regZ							
dtouchllsucond regY? extend27_offset27[regZ]		(r) regY (r) regZ						
dtouchllsucond regY? offset27[regZ]		(r) regY (r) regZ						
dtouchllsucond regY? [regZ]		(r) regY (r) regZ						
dtouchl regY[regZ]	(r) regY	(r) regZ						
dtouchl s10[regZ]	(r) regZ							
dtouchl upper27_lower10[regZ]	(r) regZ							
dzerol extend27_upper27_lower10[regZ]		(r) regZ						
dzerollsucond regY? extend27_offset27[regZ]		(r) regY (r) regZ						
dzerollsucond regY? offset27[regZ]		(r) regY (r) regZ						
dzerollsucond regY? [regZ]		(r) regY (r) regZ						
dzerol regY[regZ]	(r) regY	(r) regZ						

Instruction	ID	RR	E1	E2	E3	E4	...	E11
dzerol s10[regZ]		(r) regZ						
dzerol upper27_lower10[regZ]		(r) regZ						
fence								
i1inval								
i1invals extend27_upper27_lower10[regZ]	(r) regZ							
i1invalslsucond regY? extend27_offset27[regZ]		(r) regY (r) regZ						
i1invalslsucond regY? offset27[regZ]		(r) regY (r) regZ						
i1invalslsucond regY? [regZ]		(r) regY (r) regZ						
i1invals regY[regZ]	(r) regY	(r) regZ						
i1invals s10[regZ]	(r) regZ							
i1invals upper27_lower10[regZ]	(r) regZ							
lbsvariantdoscale regW = regY[regZ]		(r) regY (r) regZ			(w) regW			
lbsvariantlsucond regY? regW = extend27_offset27[regZ]		(r) regY (r) regZ			(w) regW			
lbsvariantlsucond regY? regW = offset27[regZ]		(r) regY (r) regZ			(w) regW			
lbsvariantlsucond regY? regW = [regZ]		(r) regY (r) regZ			(w) regW			
lbsvariant regW = extend27_upper27_lower10[regZ]		(r) regZ			(w) regW			

Instruction	ID	RR	E1	E2	E3	E4	...	E11
lbsvariant regW = s10[regZ]		(r) regZ			(w) regW			
lbsvariant regW = upper27_lower10[regZ]		(r) regZ			(w) regW			
lbzvariantdoscale regW = regY[regZ]		(r) regY (r) regZ			(w) regW			
lbzvariantlsucond regY? regW = extend27_offset27[regZ]		(r) regY (r) regZ			(w) regW			
lbzvariantlsucond regY? regW = offset27[regZ]		(r) regY (r) regZ			(w) regW			
lbzvariantlsucond regY? regW = [regZ]		(r) regY (r) regZ			(w) regW			
lbzvariant regW = extend27_upper27_lower10[regZ]		(r) regZ			(w) regW			
lbzvariant regW = s10[regZ]		(r) regZ			(w) regW			
lbzvariant regW = upper27_lower10[regZ]		(r) regZ			(w) regW			
ldvariantdoscale regW = regY[regZ]		(r) regY (r) regZ			(w) regW			
ldvariantlsucond regY? regW = extend27_offset27[regZ]		(r) regY (r) regZ			(w) regW			

Instruction	ID	RR	E1	E2	E3	E4	...	E11
ldvariantlsucond regY? regW = offset27[regZ]		(r) regY (r) regZ			(w) regW			
ldvariantlsucond regY? regW = [regZ]		(r) regY (r) regZ			(w) regW			
ldvariant regW = extend27_upper27_lower10[regZ]		(r) regZ			(w) regW			
ldvariant regW = s10[regZ]		(r) regZ			(w) regW			
ldvariant regW = upper27_lower10[regZ]		(r) regZ			(w) regW			
lhsvariantdoscale regW = regY[regZ]		(r) regY (r) regZ			(w) regW			
lhsvariantlsucond regY? regW = extend27_offset27[regZ]		(r) regY (r) regZ			(w) regW			
lhsvariantlsucond regY? regW = offset27[regZ]		(r) regY (r) regZ			(w) regW			
lhsvariantlsucond regY? regW = [regZ]		(r) regY (r) regZ			(w) regW			
lhsvariant regW = extend27_upper27_lower10[regZ]		(r) regZ			(w) regW			
lhsvariant regW = s10[regZ]		(r) regZ			(w) regW			

Instruction	ID	RR	E1	E2	E3	E4	...	E11
lhsvariant regW = upper27_lower10[regZ]		(r) regZ			(w) regW			
lhzvariantdoscale regW = regY[regZ]		(r) regY (r) regZ			(w) regW			
lhzvariantlsucond regY? regW = extend27_offset27[regZ]		(r) regY (r) regZ			(w) regW			
lhzvariantlsucond regY? regW = offset27[regZ]		(r) regY (r) regZ			(w) regW			
lhzvariantlsucond regY? regW = [regZ]		(r) regY (r) regZ			(w) regW			
lhzvariant regW = extend27_upper27_lower10[regZ]		(r) regZ			(w) regW			
lhzvariant regW = s10[regZ]		(r) regZ			(w) regW			
lhzvariant regW = upper27_lower10[regZ]		(r) regZ			(w) regW			
lovariantdoscale regN = regY[regZ]		(r) regY (r) regZ			(w) regN			
lovariantlsucond regY? regN = extend27_offset27[regZ]		(r) regY (r) regZ			(w) regN			
lovariantlsucond regY? regN = offset27[regZ]		(r) regY (r) regZ			(w) regN			

Instruction	ID	RR	E1	E2	E3	E4	...	E11
lovariantlsucond regY? regN = [regZ]		(r) regY (r) regZ			(w) regN			
lovariant regN = extend27_upper27_lower10[regZ]		(r) regZ			(w) regN			
lovariant regN = s10[regZ]		(r) regZ			(w) regN			
lovariant regN = upper27_lower10[regZ]		(r) regZ			(w) regN			
lqvariantdoscale regM = regY[regZ]		(r) regY (r) regZ			(wd) regM			
lqvariantlsucond regY? regM = extend27_offset27[regZ]		(r) regY (r) regZ			(wd) regM			
lqvariantlsucond regY? regM = offset27[regZ]		(r) regY (r) regZ			(wd) regM			
lqvariantlsucond regY? regM = [regZ]		(r) regY (r) regZ			(wd) regM			
lqvariant regM = extend27_upper27_lower10[regZ]		(r) regZ			(wd) regM			
lqvariant regM = s10[regZ]		(r) regZ			(wd) regM			
lqvariant regM = upper27_lower10[regZ]		(r) regZ			(wd) regM			

Instruction	ID	RR	E1	E2	E3	E4	...	E11
lwsvariantdoscale regW = regY[regZ]		(r) regY (r) regZ			(w) regW			
lwsvariantlsucond regY? regW = extend27_offset27[regZ]		(r) regY (r) regZ			(w) regW			
lwsvariantlsucond regY? regW = offset27[regZ]		(r) regY (r) regZ			(w) regW			
lwsvariantlsucond regY? regW = [regZ]		(r) regY (r) regZ			(w) regW			
lwsvariant regW = extend27_upper27_lower10[regZ]		(r) regZ			(w) regW			
lwsvariant regW = s10[regZ]		(r) regZ			(w) regW			
lwsvariant regW = upper27_lower10[regZ]		(r) regZ			(w) regW			
lwzvariantdoscale regW = regY[regZ]		(r) regY (r) regZ			(w) regW			
lwzvariantlsucond regY? regW = extend27_offset27[regZ]		(r) regY (r) regZ			(w) regW			
lwzvariantlsucond regY? regW = offset27[regZ]		(r) regY (r) regZ			(w) regW			

Instruction	ID	RR	E1	E2	E3	E4	...	E11
lwzvariantlsucond regY? regW = [regZ]		(r) regY (r) regZ			(w) regW			
lwzvariant regW = extend27_upper27_lower10[regZ]		(r) regZ			(w) regW			
lwzvariant regW = s10[regZ]		(r) regZ			(w) regW			
lwzvariant regW = upper27_lower10[regZ]		(r) regZ			(w) regW			
sbdoscale regY[regZ] = regT		(r) regY (r) regZ	(r) regT					
sb extend27_upper27_lower10[regZ] = regT		(r) regZ	(r) regT					
sblsucond regY? extend27_offset27[regZ] = regT		(r) regY (r) regZ	(r) regT					
sblsucond regY? offset27[regZ] = regT		(r) regY (r) regZ	(r) regT					
sblsucond regY? [regZ] = regT		(r) regY (r) regZ	(r) regT					
sb s10[regZ] = regT		(r) regZ	(r) regT					
sb upper27_lower10[regZ] = regT		(r) regZ	(r) regT					
sddoscale regY[regZ] = regT		(r) regY (r) regZ	(r) regT					
sd extend27_upper27_lower10[regZ] = regT		(r) regZ	(r) regT					
sdlsucond regY? extend27_offset27[regZ] = regT		(r) regY (r) regZ	(r) regT					
sdlsucond regY? offset27[regZ] = regT		(r) regY (r) regZ	(r) regT					
sdlsucond regY? [regZ] = regT		(r) regY (r) regZ	(r) regT					

Instruction	ID	RR	E1	E2	E3	E4	...	E11
sd s10[regZ] = regT		(r) regZ	(r) regT					
sd upper27_lower10[regZ] = regT		(r) regZ	(r) regT					
shdoscale regY[regZ] = regT		(r) regY (r) regZ	(r) regT					
sh extend27_upper27_lower10[regZ] = regT		(r) regZ	(r) regT					
shlsucond regY? extend27_offset27[regZ] = regT		(r) regY (r) regZ	(r) regT					
shlsucond regY? offset27[regZ] = regT		(r) regY (r) regZ	(r) regT					
shlsucond regY? [regZ] = regT		(r) regY (r) regZ	(r) regT					
sh s10[regZ] = regT		(r) regZ	(r) regT					
sh upper27_lower10[regZ] = regT		(r) regZ	(r) regT					
sodoscale regY[regZ] = regV		(r) regY (r) regZ	(r) regV					
so extend27_upper27_lower10[regZ] = regV		(r) regZ	(r) regV					
solsucond regY? extend27_offset27[regZ] = regV		(r) regY (r) regZ	(r) regV					
solsucond regY? offset27[regZ] = regV		(r) regY (r) regZ	(r) regV					
solsucond regY? [regZ] = regV		(r) regY (r) regZ	(r) regV					
so s10[regZ] = regV		(r) regZ	(r) regV					
so upper27_lower10[regZ] = regV		(r) regZ	(r) regV					
sqdoscale regY[regZ] = regU		(r) regY (r) regZ	(rd) regU					
sq extend27_upper27_lower10[regZ] = regU		(r) regZ	(rd) regU					
sqlsucond regY? extend27_offset27[regZ] = regU		(r) regY (r) regZ	(rd) regU					

Instruction	ID	RR	E1	E2	E3	E4	...	E11
sqlsucond regY? offset27[regZ] = regU		(r) regY (r) regZ	(rd) regU					
sqlsucond regY? [regZ] = regU		(r) regY (r) regZ	(rd) regU					
sq s10[regZ] = regU		(r) regZ	(rd) regU					
sq upper27_lower10[regZ] = regU		(r) regZ	(rd) regU					
swdoscale regY[regZ] = regT		(r) regY (r) regZ	(r) regT					
sw extend27_upper27_lower10[regZ] = regT		(r) regZ	(r) regT					
swlsucond regY? extend27_offset27[regZ] = regT		(r) regY (r) regZ	(r) regT					
swlsucond regY? offset27[regZ] = regT		(r) regY (r) regZ	(r) regT					
swlsucond regY? [regZ] = regT		(r) regY (r) regZ	(r) regT					
sw s10[regZ] = regT		(r) regZ	(r) regT					
sw upper27_lower10[regZ] = regT		(r) regZ	(r) regT					
xlospeculatedoscale regG = regY[regZ]		(r) regY (r) regZ						
xlospeculatelsucondqindex regY? regGq = extend27_offset27[regZ]		(r) regY (r) regZ			(r) regGq (r) regGq (r) regGq (r) regGq			

Instruction	ID	RR	E1	E2	E3	E4	...	E11
xlospeculatelsucondqindex regY? regGq = offset27[regZ]		(r) regY (r) regZ			(r) regGq (r) regGq (r) regGq (r) regGq			
xlospeculatelsucondqindex regY? regGq = [regZ]		(r) regY (r) regZ			(r) regGq (r) regGq (r) regGq (r) regGq			
xlospeculatelsucond regY? regG = extend27_offset27[regZ]		(r) regY (r) regZ						
xlospeculatelsucond regY? regG = offset27[regZ]		(r) regY (r) regZ						
xlospeculatelsucond regY? regG = [regZ]		(r) regY (r) regZ						

Instruction	ID	RR	E1	E2	E3	E4	...	E11
xlospeculateqindexdoscale regGq = regY[regZ]		(r) regY (r) regZ			(r) regGq (r) regGq (r) regGq (r) regGq			
xlospeculateqindex regGq = extend27_upper27_lower10[regZ]		(r) regZ			(r) regGq (r) regGq (r) regGq (r) regGq			
xlospeculateqindex regGq = s10[regZ]		(r) regZ			(r) regGq (r) regGq (r) regGq (r) regGq			

KETD:

Instruction	ID	RR	E1	E2	E3	E4	...	E11
xlospeculateqindex regGq = upper27_lower10[regZ]		(r) regZ			(r) regGq (r) regGq (r) regGq (r) regGq			
xlospeculate regG = extend27_upper27_lower10[regZ]		(r) regZ						
xlospeculate regG = s10[regZ]		(r) regZ						
xlospeculate regG = upper27_lower10[regZ]		(r) regZ						
xsodoscale regY[regZ] = regE		(r) regY (r) regZ	(r) regE					
xso extend27_upper27_lower10[regZ] = regE		(r) regZ	(r) regE					
xsolsucond regY? extend27_offset27[regZ] = regE		(r) regY (r) regZ	(r) regE					
xsolsucond regY? offset27[regZ] = regE		(r) regY (r) regZ	(r) regE					
xsolsucond regY? [regZ] = regE		(r) regY (r) regZ	(r) regE					
xso s10[regZ] = regE		(r) regZ	(r) regE					
xso upper27_lower10[regZ] = regE		(r) regZ	(r) regE					

8.4 BCU instructions

Instruction	ID	RR	E1	E2	E3	E4
aligno regN = regBe, regCo, byteshift		(r) regBe (r) regCo			(w) regN	
aligno regN = regBe, regCo, regZ	(r) regZ	(r) regBe (r) regCo			(w) regN	
aligno regN = regBo, regCe, byteshift		(r) regBo (r) regCe			(w) regN	
aligno regN = regBo, regCe, regZ	(r) regZ	(r) regBo (r) regCe			(w) regN	
alignv regA = regBe, regCo, byteshift		(r) regBe (r) regCo			(w) regA	
alignv regA = regBe, regCo, regZ	(r) regZ	(r) regBe (r) regCo			(w) regA	
alignv regA = regBo, regCe, byteshift		(r) regBo (r) regCe			(w) regA	
alignv regA = regBo, regCe, regZ	(r) regZ	(r) regBo (r) regCe			(w) regA	
await						
barrier						
call pcrel27						

Instruction	ID	RR	E1	E2	E3	E4
cbbranchcond regZ? pcrel17	(r) regZ					
errop						
get regZ = sysS2		(r) sysS2	(w) regZ			
get regZ = sysS3		(r) sysS3	(w) regZ			
goto pcrel27						
icall regZ	(r) regZ					
iget regZ	(r) regZ		(w) regZ			
igoto regZ	(r) regZ					
loopdo regZ, pcrel17	(r) regZ					
ret						
rfe						
rswap regZ = sysAlone		(r) regZ (r) sysAlone	(w) regZ (w) sysAlone			
rswap regZ = sysS3		(r) regZ (r) sysS3	(w) regZ (w) sysS3			
rswap regZ = sysS4		(r) regZ (r) sysS4	(w) regZ (w) sysS4			
scall regZ	(r) regZ					

Instruction	ID	RR	E1	E2	E3	E4
scall sysnumber						
set sysAlone = regZ	(r) regZ				(w) sysAlone	
set RA = regZ	(r) regZ				(w) RA	
set sysT3 = regZ	(r) regZ				(w) sysT3	
set sysT4 = regZ	(r) regZ				(w) sysT4	
sleep						
stop						
syncgroup regZ	(r) regZ					
tlbdinval						
tlbiinval						
tlbprobe						
tlbread						
tlbwrite						
waitit regZ	(r) regZ	(w) regZ				
wfxl sysAlone, regZ	(r) regZ					(w) sysAlone
wfxl sysT2, regZ	(r) regZ					(w) sysT2
wfxl sysT4, regZ	(r) regZ					(w) sysT4

Instruction	ID	RR	E1	E2	E3	E4
wfxm sysAlone, regZ	(r) regZ					(w) sysAlone
wfxm sysT2, regZ	(r) regZ					(w) sysT2
wfxm sysT4, regZ	(r) regZ					(w) sysT4
xcopyo regA = regBe		(r) regBe			(w) regA	
xcopyo regA = regBo		(r) regBo			(w) regA	
xmovefo regN = regBe		(r) regBe			(w) regN	
xmovefo regN = regBo		(r) regBo			(w) regN	

9 Instructions index

This section is an alphabetical index of all the instructions, giving, for each of them, its associated EXU (i.e. the EXU on which this instruction executes), the (simplified) reservation class it belongs to and linking to its operands'stages description in the relevant table of section 8.

The reservation classes that are used here are the real ones (as defined in the Instruction Reservation Tables of the architecture manual) except that the last part of the names (like ".X") have been trimmed as it relates solely to the presence or the absence of immediate extensions, whereas our table only deals with mnemonics, not complete formats, so it would not make sense. These pieces of reservation information are very interesting in that they tell us:

information	reservation pattern
alu instruction is supported in double TINY ALU unit	TINY
instruction uses the shared double read port RPD2	*_ACC_*
instruction uses the carry	*_ODD_*
instruction must be issued alone in their bundle	ALL

These point have to be taken into account to form correct bundles when writing assembly. Of course, the bundling of instructions is further constrained by the issue ressource (8 syllables maximum per bundle) as detailed in the Instruction Reservation Tables section of the architecture manual.

Mnemonic	**Execution Unit**	**Reservation**
abdd	ALU	ALU_LITE
abdhq	ALU	ALU_LITE
abdw	ALU	ALU_LITE
abdwp	ALU	ALU_LITE
absd	ALU	ALU_LITE
abshq	ALU	ALU_LITE
absw	ALU	ALU_LITE
abswp	ALU	ALU_LITE
acswapd	LSU	LSU_AUXR_AUXW
acswapw	LSU	LSU_AUXR_AUXW
addcd	ALU	ALU_FULL
addcd.*i*	ALU	ALU_FULL
addd	ALU	ALU_TINY
addhcp.*c*	ALU	ALU_LITE
addhq	ALU	ALU_TINY
addsd	ALU	ALU_LITE
addshq	ALU	ALU_LITE
addsw	ALU	ALU_LITE
addswp	ALU	ALU_LITE

Mnemonic	Execution Unit	Reservation
adduwd	ALU	ALU_LITE
addw	ALU	ALU_TINY
addwc.*c*	ALU	ALU_LITE
addwd	ALU	ALU_LITE
addwp	ALU	ALU_TINY
addx16d	ALU	ALU_LITE
addx16hq	ALU	ALU_LITE
addx16uwd	ALU	ALU_LITE
addx16w	ALU	ALU_LITE
addx16wd	ALU	ALU_LITE
addx16wp	ALU	ALU_LITE
addx2d	ALU	ALU_LITE
addx2hq	ALU	ALU_LITE
addx2uwd	ALU	ALU_LITE
addx2w	ALU	ALU_LITE
addx2wd	ALU	ALU_LITE
addx2wp	ALU	ALU_LITE
addx4d	ALU	ALU_LITE
addx4hq	ALU	ALU_LITE
addx4uwd	ALU	ALU_LITE
addx4w	ALU	ALU_LITE
addx4wd	ALU	ALU_LITE
addx4wp	ALU	ALU_LITE
addx8d	ALU	ALU_LITE
addx8hq	ALU	ALU_LITE
addx8uwd	ALU	ALU_LITE
addx8w	ALU	ALU_LITE
addx8wd	ALU	ALU_LITE
addx8wp	ALU	ALU_LITE
aladdd	LSU	LSU_AUXR_AUXW
aladdw	LSU	LSU_AUXR_AUXW
alclrd	LSU	LSU_AUXW
alclrw	LSU	LSU_AUXW
aligno	BCU	BCU_TINY_AUXW_CRRP
alignv	BCU	BCU_CRRP_CRWL_CRWH
andd	ALU	ALU_TINY
andnd	ALU	ALU_TINY
andnw	ALU	ALU_TINY
andw	ALU	ALU_TINY
avghq	ALU	ALU_LITE
avgrhq	ALU	ALU_LITE
avgruhq	ALU	ALU_LITE
avgruw	ALU	ALU_LITE

Mnemonic	Execution Unit	Reservation
avgruwp	ALU	ALU_LITE
avgrw	ALU	ALU_LITE
avgrwp	ALU	ALU_LITE
avguhq	ALU	ALU_LITE
avguw	ALU	ALU_LITE
avguwp	ALU	ALU_LITE
avgw	ALU	ALU_LITE
avgwp	ALU	ALU_LITE
await	BCU	ALL
barrier	BCU	ALL
call	BCU	BCU
cb	BCU	BCU
cbsd	ALU	ALU_LITE
cbsw	ALU	ALU_LITE
cbswp	ALU	ALU_LITE
clrf	ALU	ALU_LITE
clsd	ALU	ALU_LITE
clsw	ALU	ALU_LITE
clswp	ALU	ALU_LITE
clzd	ALU	ALU_LITE
clzw	ALU	ALU_LITE
clzwp	ALU	ALU_LITE
cmoved	ALU	ALU_LITE
cmovehq	ALU	ALU_LITE
cmovewp	ALU	ALU_LITE
cmuldt	MAU	MAU
cmulghxdt	MAU	MAU_AUXR
cmulglxdt	MAU	MAU_AUXR
cmulgmxdt	MAU	MAU_AUXR
cmulxdt	MAU	MAU_AUXR
compd	ALU	ALU_TINY
compnhq	ALU	ALU_TINY
compnwp	ALU	ALU_TINY
compuwd	ALU	ALU_LITE
compw	ALU	ALU_TINY
compwd	ALU	ALU_LITE
convdhv0	EXT	EXT
convdhv1	EXT	EXT
convwbv0	EXT	EXT
convwbv1	EXT	EXT
convwbv2	EXT	EXT
convwbv3	EXT	EXT
copyd	ALU	ALU_TINY

Mnemonic	Execution Unit	Reservation
copyo	LSU	LSU_AUXR_AUXW
copyq	MAU	MAU
copyw	ALU	ALU_TINY
crcbellw	MAU	MAU_AUXR
crcbelmw	MAU	MAU_AUXR
crclellw	MAU	MAU_AUXR
crclelmw	MAU	MAU_AUXR
ctzd	ALU	ALU_LITE
ctzw	ALU	ALU_LITE
ctzwp	ALU	ALU_LITE
d1inval	LSU	LSU
dinvall	LSU	LSU
dot2suwd	MAU	MAU
dot2suwdp	MAU	MAU_AUXR
dot2uwd	MAU	MAU
dot2uwdp	MAU	MAU_AUXR
dot2w	MAU	MAU
dot2wd	MAU	MAU
dot2wdp	MAU	MAU_AUXR
dot2wzp	MAU	MAU_AUXR
dtouchl	LSU	LSU
dzerol	LSU	LSU
eord	ALU	ALU_TINY
eorw	ALU	ALU_TINY
errop	BCU	ALL
extfs	ALU	ALU_LITE
extfz	ALU	ALU_LITE
fabsd	ALU	ALU_LITE
fabshq	ALU	ALU_LITE
fabsw	ALU	ALU_LITE
fabswp	ALU	ALU_LITE
faddd	MAU	MAU
fadddc	MAU	MAU_AUXR
fadddc.*c*	MAU	MAU_AUXR
fadddp	MAU	MAU_AUXR
faddhq	MAU	MAU
faddw	MAU	MAU
faddwc	MAU	MAU
faddwc.*c*	MAU	MAU
faddwcp	MAU	MAU_AUXR
faddwcp.*c*	MAU	MAU_AUXR
faddwp	MAU	MAU
faddwq	MAU	MAU_AUXR

Mnemonic	Execution Unit	Reservation
fcdivd	ALU	ALU_LITE
fcdivw	ALU	ALU_LITE
fcdivwp	ALU	ALU_LITE
fcompd	ALU	ALU_LITE
fcompnhq	ALU	ALU_LITE
fcompnwp	ALU	ALU_LITE
fcompw	ALU	ALU_LITE
fdot2w	MAU	MAU
fdot2wd	MAU	MAU
fdot2wdp	MAU	MAU_AUXR
fdot2wzp	MAU	MAU_AUXR
fence	LSU	LSU
ffmad	MAU	MAU_AUXR
ffmahq	MAU	MAU_AUXR
ffmahw	MAU	MAU_AUXR
ffmahwq	MAU	MAU_AUXR
ffmaw	MAU	MAU_AUXR
ffmawd	MAU	MAU_AUXR
ffmawdp	MAU	MAU_AUXR
ffmawp	MAU	MAU_AUXR
ffmsd	MAU	MAU_AUXR
ffmshq	MAU	MAU_AUXR
ffmshw	MAU	MAU_AUXR
ffmshwq	MAU	MAU_AUXR
ffmsw	MAU	MAU_AUXR
ffmswd	MAU	MAU_AUXR
ffmswdp	MAU	MAU_AUXR
ffmswp	MAU	MAU_AUXR
fixedd	MAU	MAU
fixedud	MAU	MAU
fixeduw	MAU	MAU
fixeduwp	MAU	MAU
fixedw	MAU	MAU
fixedwp	MAU	MAU
floatd	MAU	MAU
floatud	MAU	MAU
floatuw	MAU	MAU
floatuwp	MAU	MAU
floatw	MAU	MAU
floatwp	MAU	MAU
fmaxd	ALU	ALU_LITE
fmaxhq	ALU	ALU_LITE
fmaxw	ALU	ALU_LITE

Mnemonic	Execution Unit	Reservation
fmaxwp	ALU	ALU_LITE
fmind	ALU	ALU_LITE
fminhq	ALU	ALU_LITE
fminw	ALU	ALU_LITE
fminwp	ALU	ALU_LITE
fmm212w	MAU	MAU
fmma212w	MAU	MAU_AUXR
fmma242hw0	EXT	EXT
fmma242hw1	EXT	EXT
fmma242hw2	EXT	EXT
fmma242hw3	EXT	EXT
fmms212w	MAU	MAU_AUXR
fmuld	MAU	MAU
fmulhq	MAU	MAU
fmulhw	MAU	MAU
fmulhwq	MAU	MAU
fmulw	MAU	MAU
fmulwc	MAU	MAU
fmulwc.*c*	MAU	MAU
fmulwd	MAU	MAU
fmulwdc	MAU	MAU
fmulwdc.*c*	MAU	MAU
fmulwdp	MAU	MAU
fmulwp	MAU	MAU
fmulwq	MAU	MAU_AUXR
fnarrow44wh	EXT	EXT
fnarrowdw	ALU	ALU_FULL
fnarrowdwp	ALU	ALU_FULL
fnarrowwh	ALU	ALU_LITE
fnarrowwhq	ALU	ALU_LITE
fnegd	ALU	ALU_LITE
fneghq	ALU	ALU_LITE
fnegw	ALU	ALU_LITE
fnegwp	ALU	ALU_LITE
frecw	ALU	ALU_FULL
frsrw	ALU	ALU_FULL
fsbfd	MAU	MAU
fsbfdc	MAU	MAU_AUXR
fsbfdc.*c*	MAU	MAU_AUXR
fsbfdp	MAU	MAU_AUXR
fsbfhq	MAU	MAU
fsbfw	MAU	MAU
fsbfwc	MAU	MAU

Mnemonic	Execution Unit	Reservation
fsbfwc.*c*	MAU	MAU
fsbfwcp	MAU	MAU_AUXR
fsbfwcp.*c*	MAU	MAU_AUXR
fsbfwp	MAU	MAU
fsbfwq	MAU	MAU_AUXR
fscalewv	EXT	EXT
fsdivd	ALU	ALU_LITE
fsdivw	ALU	ALU_LITE
fsdivwp	ALU	ALU_LITE
fsrecd	ALU	ALU_LITE
fsrecw	ALU	ALU_LITE
fsrecwp	ALU	ALU_LITE
fsrsrd	ALU	ALU_LITE
fsrsrw	ALU	ALU_LITE
fsrsrwp	ALU	ALU_LITE
fwidenlhw	ALU	ALU_LITE
fwidenlhwp	ALU	ALU_LITE
fwidenlwd	ALU	ALU_LITE
fwidenmhw	ALU	ALU_LITE
fwidenmhwp	ALU	ALU_LITE
fwidenmwd	ALU	ALU_LITE
get	BCU	BCU_TINY_TINY_MAU_XNOP
goto	BCU	BCU
i1inval	LSU	LSU
i1invals	LSU	LSU
icall	BCU	BCU
iget	BCU	BCU_TINY_TINY_MAU_XNOP
igoto	BCU	BCU
insf	ALU	ALU_LITE
iord	ALU	ALU_TINY
iornd	ALU	ALU_TINY
iornw	ALU	ALU_TINY
iorw	ALU	ALU_TINY
landd	ALU	ALU_LITE
landhq	ALU	ALU_LITE
landw	ALU	ALU_LITE
landwp	ALU	ALU_LITE
lbs	LSU	LSU_AUXW
lbz	LSU	LSU_AUXW
ld	LSU	LSU_AUXW
lhs	LSU	LSU_AUXW
lhz	LSU	LSU_AUXW
liord	ALU	ALU_LITE

Mnemonic	Execution Unit	Reservation
liorhq	ALU	ALU_LITE
liorw	ALU	ALU_LITE
liorwp	ALU	ALU_LITE
lnandd	ALU	ALU_LITE
lnandhq	ALU	ALU_LITE
lnandw	ALU	ALU_LITE
lnandwp	ALU	ALU_LITE
lniord	ALU	ALU_LITE
lniorhq	ALU	ALU_LITE
lniorw	ALU	ALU_LITE
lniorwp	ALU	ALU_LITE
lnord	ALU	ALU_LITE
lnorhq	ALU	ALU_LITE
lnorw	ALU	ALU_LITE
lnorwp	ALU	ALU_LITE
lo	LSU	LSU_AUXW
loopdo	BCU	ALL
lord	ALU	ALU_LITE
lorhq	ALU	ALU_LITE
lorw	ALU	ALU_LITE
lorwp	ALU	ALU_LITE
lq	LSU	LSU_AUXW
lws	LSU	LSU_AUXW
lwz	LSU	LSU_AUXW
maddd	MAU	MAU_AUXR
madddt	MAU	MAU_AUXR
maddhq	MAU	MAU_AUXR
maddhwq	MAU	MAU_AUXR
maddsudt	MAU	MAU_AUXR
maddsuhwq	MAU	MAU_AUXR
maddsuwd	MAU	MAU_AUXR
maddsuwdp	MAU	MAU_AUXR
maddudt	MAU	MAU_AUXR
madduhwq	MAU	MAU_AUXR
madduwd	MAU	MAU_AUXR
madduwdp	MAU	MAU_AUXR
madduzdt	MAU	MAU_AUXR
maddw	MAU	MAU_AUXR
maddwd	MAU	MAU_AUXR
maddwdp	MAU	MAU_AUXR
maddwp	MAU	MAU_AUXR
make	ALU	ALU_TINY
maxd	ALU	ALU_TINY

Mnemonic	Execution Unit	Reservation
maxhq	ALU	ALU_TINY
maxud	ALU	ALU_TINY
maxuhq	ALU	ALU_TINY
maxuw	ALU	ALU_TINY
maxuwp	ALU	ALU_TINY
maxw	ALU	ALU_TINY
maxwp	ALU	ALU_TINY
mind	ALU	ALU_TINY
minhq	ALU	ALU_TINY
minud	ALU	ALU_TINY
minuhq	ALU	ALU_TINY
minuw	ALU	ALU_TINY
minuwp	ALU	ALU_TINY
minw	ALU	ALU_TINY
minwp	ALU	ALU_TINY
mm212w	MAU	MAU
mma212w	MAU	MAU_AUXR
mma444hbd0	EXT	EXT
mma444hbd1	EXT	EXT
mma444hd	EXT	EXT
mma444suhbd0	EXT	EXT
mma444suhbd1	EXT	EXT
mma444suhd	EXT	EXT
mma444uhbd0	EXT	EXT
mma444uhbd1	EXT	EXT
mma444uhd	EXT	EXT
mma444ushbd0	EXT	EXT
mma444ushbd1	EXT	EXT
mma444ushd	EXT	EXT
mms212w	MAU	MAU_AUXR
movetq	ALU	ALU_LITE_CRWH
msbfd	MAU	MAU_AUXR
msbfdt	MAU	MAU_AUXR
msbfhq	MAU	MAU_AUXR
msbfhwq	MAU	MAU_AUXR
msbfsudt	MAU	MAU_AUXR
msbfsuhwq	MAU	MAU_AUXR
msbfsuwd	MAU	MAU_AUXR
msbfsuwdp	MAU	MAU_AUXR
msbfudt	MAU	MAU_AUXR
msbfuhwq	MAU	MAU_AUXR
msbfuwd	MAU	MAU_AUXR
msbfuwdp	MAU	MAU_AUXR

Mnemonic	Execution Unit	Reservation
msbfuzdt	MAU	MAU_AUXR
msbfw	MAU	MAU_AUXR
msbfwd	MAU	MAU_AUXR
msbfwdp	MAU	MAU_AUXR
msbfwp	MAU	MAU_AUXR
muld	MAU	MAU
muldt	MAU	MAU
mulhq	MAU	MAU
mulhwq	MAU	MAU
mulsudt	MAU	MAU
mulsuhwq	MAU	MAU
mulsuwd	MAU	MAU
mulsuwdp	MAU	MAU
muludt	MAU	MAU
muluhwq	MAU	MAU
muluwd	MAU	MAU
muluwdp	MAU	MAU
mulw	MAU	MAU
mulwc	MAU	MAU
mulwc.*c*	MAU	MAU
mulwd	MAU	MAU
mulwdc	MAU	MAU
mulwdc.*c*	MAU	MAU
mulwdp	MAU	MAU
mulwp	MAU	MAU
mulwq	MAU	MAU_AUXR
nandd	ALU	ALU_TINY
nandw	ALU	ALU_TINY
negd	ALU	ALU_TINY
neghq	ALU	ALU_TINY
negw	ALU	ALU_TINY
negwp	ALU	ALU_TINY
neord	ALU	ALU_TINY
neorw	ALU	ALU_TINY
niord	ALU	ALU_TINY
niorw	ALU	ALU_TINY
nop	ALU	ALU_NOP
nord	ALU	ALU_TINY
norw	ALU	ALU_TINY
notd	ALU	ALU_TINY
notw	ALU	ALU_TINY
nxord	ALU	ALU_TINY
nxorw	ALU	ALU_TINY

Mnemonic	Execution Unit	Reservation
ord	ALU	ALU_TINY
ornd	ALU	ALU_TINY
ornw	ALU	ALU_TINY
orw	ALU	ALU_TINY
pcrel	ALU	ALU_FULL
ret	BCU	BCU
rfe	BCU	ALL
rolw	ALU	ALU_LITE
rolwps	ALU	ALU_LITE
rorw	ALU	ALU_LITE
rorwps	ALU	ALU_LITE
rswap	BCU	BCU_TINY_TINY_MAU_XNOP
satd	ALU	ALU_LITE
satdh	ALU	ALU_LITE
satdw	ALU	ALU_LITE
sb	LSU	LSU_AUXR
sbfcd	ALU	ALU_FULL
sbfcd.*i*	ALU	ALU_FULL
sbfd	ALU	ALU_TINY
sbfhcp.*c*	ALU	ALU_LITE
sbfhq	ALU	ALU_TINY
sbfsd	ALU	ALU_LITE
sbfshq	ALU	ALU_LITE
sbfsw	ALU	ALU_LITE
sbfswp	ALU	ALU_LITE
sbfuwd	ALU	ALU_LITE
sbfw	ALU	ALU_TINY
sbfwc.*c*	ALU	ALU_LITE
sbfwd	ALU	ALU_LITE
sbfwp	ALU	ALU_TINY
sbfx16d	ALU	ALU_LITE
sbfx16hq	ALU	ALU_LITE
sbfx16uwd	ALU	ALU_LITE
sbfx16w	ALU	ALU_LITE
sbfx16wd	ALU	ALU_LITE
sbfx16wp	ALU	ALU_LITE
sbfx2d	ALU	ALU_LITE
sbfx2hq	ALU	ALU_LITE
sbfx2uwd	ALU	ALU_LITE
sbfx2w	ALU	ALU_LITE
sbfx2wd	ALU	ALU_LITE
sbfx2wp	ALU	ALU_LITE
sbfx4d	ALU	ALU_LITE

Mnemonic	Execution Unit	Reservation
sbfx4hq	ALU	ALU_LITE
sbfx4uwd	ALU	ALU_LITE
sbfx4w	ALU	ALU_LITE
sbfx4wd	ALU	ALU_LITE
sbfx4wp	ALU	ALU_LITE
sbfx8d	ALU	ALU_LITE
sbfx8hq	ALU	ALU_LITE
sbfx8uwd	ALU	ALU_LITE
sbfx8w	ALU	ALU_LITE
sbfx8wd	ALU	ALU_LITE
sbfx8wp	ALU	ALU_LITE
sbmm8	ALU	ALU_LITE
sbmm8d	ALU	ALU_LITE
sbmmt8	ALU	ALU_LITE
sbmmt8d	ALU	ALU_LITE
scall	BCU	ALL
sd	LSU	LSU_AUXR
set	BCU	BCU
sh	LSU	LSU_AUXR
sleep	BCU	ALL
slld	ALU	ALU_TINY
sllhqs	ALU	ALU_LITE
sllw	ALU	ALU_TINY
sllwps	ALU	ALU_LITE
slsd	ALU	ALU_LITE
slshqs	ALU	ALU_LITE
slsw	ALU	ALU_LITE
slswps	ALU	ALU_LITE
so	LSU	LSU_AUXR
sq	LSU	LSU_AUXR
srad	ALU	ALU_TINY
srahqs	ALU	ALU_LITE
sraw	ALU	ALU_TINY
srawps	ALU	ALU_LITE
srld	ALU	ALU_TINY
srlhqs	ALU	ALU_LITE
srlw	ALU	ALU_TINY
srlwps	ALU	ALU_LITE
srsd	ALU	ALU_LITE
srshqs	ALU	ALU_LITE
srsw	ALU	ALU_LITE
srswps	ALU	ALU_LITE
stop	BCU	ALL

Mnemonic	Execution Unit	Reservation
stsud	ALU	ALU_LITE
stsuw	ALU	ALU_LITE
sw	LSU	LSU_AUXR
sxbd	ALU	ALU_LITE
sxhd	ALU	ALU_LITE
sxlbhq	ALU	ALU_LITE
sxlhwp	ALU	ALU_LITE
sxmbhq	ALU	ALU_LITE
sxmhwp	ALU	ALU_LITE
sxwd	ALU	ALU_LITE
syncgroup	BCU	BCU
tlbdinval	BCU	ALL
tlbiinval	BCU	ALL
tlbprobe	BCU	ALL
tlbread	BCU	ALL
tlbwrite	BCU	ALL
waitit	BCU	BCU_TINY_TINY_MAU_XNOP
wfxl	BCU	BCU
wfxm	BCU	BCU
xcopyo	BCU	BCU_CRRP_CRWL_CRWH
xlo	LSU	LSU
xmma484bw	EXT	EXT
xmma484subw	EXT	EXT
xmma484ubw	EXT	EXT
xmma484usbw	EXT	EXT
xmovefo	BCU	BCU_TINY_AUXW_CRRP
xmovetq	ALU	ALU_LITE_CRWH
xmt44d	EXT	EXT
xord	ALU	ALU_TINY
xorw	ALU	ALU_TINY
xso	LSU	LSU_CRRP
zxbd	ALU	ALU_TINY
zxhd	ALU	ALU_LITE
zxwd	ALU	ALU_TINY