SECTION 2
CPU32X

This document describes the modifications of the CPU32 to create the CPU32X.

2.1 Features

The CPU32X is greater than two times faster at the same external clock rate than the CPU32 using similar speed memory devices. This performance improvement is obtained by using burst mode memory for instruction fetches, an instruction queue to buffer the burst instruction fetches, one clock operand (data) accesses on the fast access bus and by increasing the speed of the CPU.

NOTE

All references to clocks are to the external clock.

The CPU32X and CPU32 instruction sets are identical. Interrupts, breakpoints and bus errors function exactly the same in the CPU32X as they do in the CPU32 except with reduced response time. Retry, relinquish and retry (RRT), and late bus error are not supported by the CPU32X. Please refer to the ***CPU32 Reference Manual*** **(CPU32RM/AD)**, for the instruction set, interrupt operation, breakpoint operation, bus error operation and programmer's model.

The CPU32X burst mode bus protocol is an extension of the intermodule bus (IMB) protocol. The CPU32X will support all of the existing and future IMB peripherals. Burst mode IMB program memory (EEPROM, flash EPROM, ROM) can be implemented on the IMB to support the CPU32X. The external burst IMB protocol is supported by the burst mode system integration module (BIM).

The fast access bus is connected to the fast access static RAM (FASRAM) and increases the data fetch bandwidth of the CPU32X by reducing the minimum data access time to one clock. In addition the fast access bus separates data accesses from program accesses such that if the data being accessed is in the FASRAM the data will be returned to the CPU in one clock and the IMB burst mode program fetch can continue without interruption.

2.2 CPU32X System Architecture

The recommended system architecture of the CPU32X includes a fast access standby random access memory (FASRAM) and requires a burst mode interface module (BIM). Refer to **Figure 2-1** for the block diagram of a CPU32X-based device.

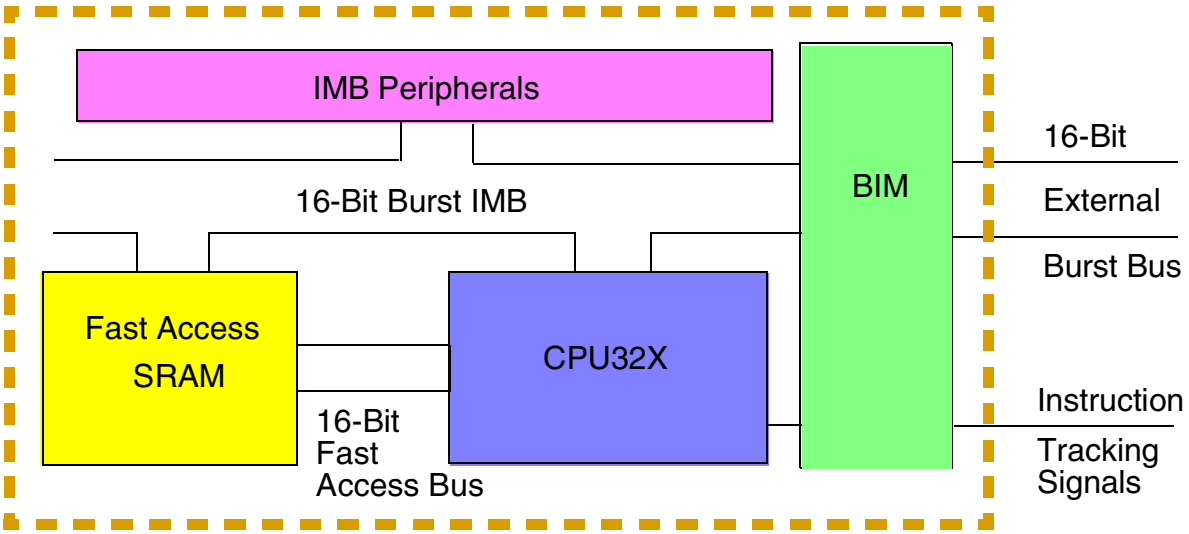


Figure 2-1 System Architecture

2.3 Optimizing System Performance

Data accesses to the fast access SRAM optimize the performance of the CPU32X system therefore the stack and the most frequently used variables should be located in the FASRAM. Accesses in the FASRAM are one clock while accesses outside the FASRAM normally require two clocks plus the number of clocks required to run the IMB or external access. For example the access time of a two-clock IMB SRAM would be four clocks total (two clocks of internal delay plus two clocks for IMB SRAM access). The normal total access time for any IMB based memory, IMB based peripheral or external device is four clocks. (Two clocks delay from the CPU32X and two clocks for the fastest possible bus cycle). If IMB or external bus cycles are longer than two clocks the bus cycle will be extended. If for some reason the IMB is idle then the minimum number of clocks for a non-FASRAM access is three clocks. The IMB can be idle due to a full prefetch queue or for other reasons. Refer to **Table 2-2** for an example of a data access to the FASRAM. Refer to **Table 2-3** for an example of a data access not in the FASRAM.

2.4 Instruction Execution

This section describes instruction execution on the CPU32X. External clock cycles are used to provide as accurate as possible operation and timing of the instruction examples. Because exact execution time for an instruction or operation depends on independently scheduled resources, on memory speeds, and on other variables these diagrams may not exactly match the CPU32X operation in every application.

2.4.1 Resource Scheduling

The CPU32X contains several independently scheduled resources. The organization of these resources within the CPU32X is shown in **Figure 2-2**. Some variation in

instruction execution timing results from concurrent resource utilization. Due to these concurrent resources exact instruction execution times are difficult to predict until the instruction sequence is actually executed. Identical sequences of instructions and memory accesses will not vary in execution time or bus access pattern from one execution of the identical sequence to another.

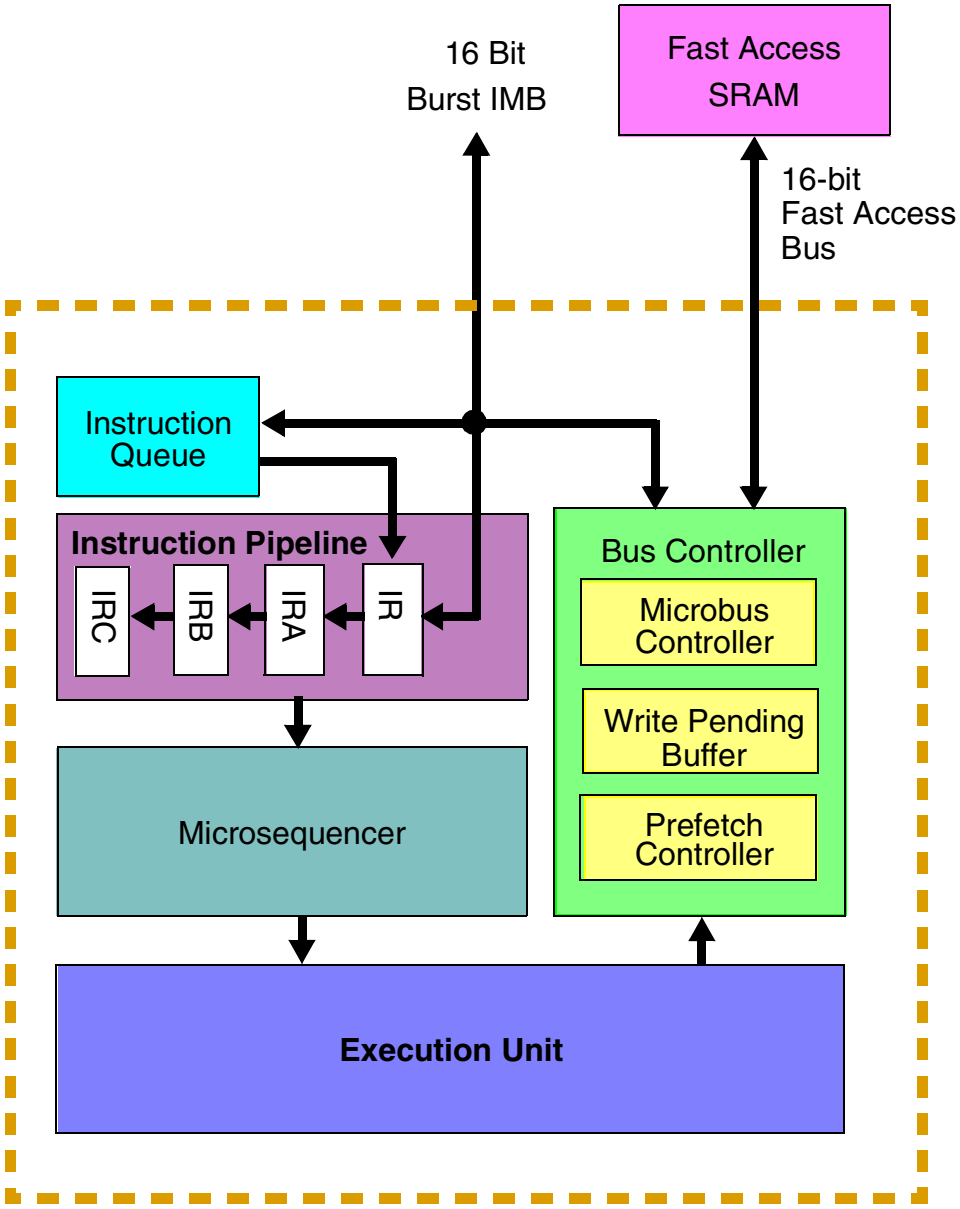


Figure 2-2 CPU32X Internal Architecture

2.4.2 Microsequencer

The microsequencer either executes microinstructions or awaits completion of accesses necessary to continue microcode execution. The microsequencer supervises the bus controller, instruction execution, and internal processor operations such

as the calculation of an effective address and the setting of condition codes. It also initiates instruction word prefetches after a change of flow and controls validation of instruction words in the instruction pipeline. The microsequencer is the same on the CPU32 as on the CPU32X.

2.4.3 Instruction Pipeline

The CPU32X contains a four-word instruction pipeline where instruction opcodes are decoded. The pipeline operates in two stages: IRA and IRC. Each stage of the pipeline is initially filled under microsequencer control and subsequently refilled by the prefetch controller as it empties.

The IR word of the instruction pipeline is a buffer. IR receives instructions from either the instruction queue or directly from the intermodule bus. This register holds the instruction word until it is emptied by IRA. Instruction words (instruction operation words and all extension words) are decoded at stage IRA. IRB is an intermediate holding register for IRC and is filled from IRA when IRC empties. IRC is filled from IRB. Residual decoding and execution take place in stage IRC.

The instruction pipeline registers IRA, IRB, IRC are the same as on the CPU32. The IR registers and the prefetch equations will require modification to support the CPU32X instruction queue and burst mode bus.

2.4.4 Instruction Queue

The instruction queue is a six word first-in-first-out buffer. The instruction queue is filled by the prefetch controller when the IR word of the pipeline is full. When IR empties, the instruction queue provides the next instruction word or if the instruction queue is empty, the instruction word will come directly off the IMB and into IR. The instruction queue is dual ported such that the prefetch controller can fill the queue as the pipe pulls instructions or extension words out of the queue.

Each word in the queue has a bus error and breakpoint status bit which indicates that the word in that stage was loaded with an instruction or extension word from a bus cycle that encountered either a bus error or a breakpoint. This status is either transferred to the pipe when the word is used in the pipe or they are flushed on a change of flow.

The instruction queue is new on the CPU32X.

2.5 Bus Controller Resources

The bus controller consists of the instruction prefetch controller, the write-pending buffer and the microbus controller. These resources transact all reads, writes, and instruction prefetches required for instruction execution.

The bus controller and microsequencer operate concurrently. The bus controller schedules bus cycles to the IMB and the FASRAM. The bus controller can schedule an IMB prefetch in parallel with a FASRAM access or an IMB access. While the bus

controller is running bus cycles the microsequencer controls effective address calculation or sets condition codes.

The microsequencer can also request a bus cycle. If the data resides in the FASRAM the bus controller completes the bus cycle to the FASRAM, see **Table 2-2**. If the data is not in the FASRAM the bus controller queues the data cycle to run on the IMB. The bus controller then terminates any prefetches in progress and runs the cycle when the current cycle is complete. See **Table 2-3** for more information. Once the bus cycle completes, data is returned to the execution unit. If the first word of a long-word access is not in the FASRAM the microsequencer will run the second word of the long-word access on the IMB bypassing the FASRAM access for the second word. If the second word is in the FASRAM the data will be accessed over the IMB. This is done to improve performance of long-word accesses on the IMB.

2.5.1 Prefetch Controller

The instruction prefetch controller receives an initial request from the microsequencer to initiate burst mode instruction prefetching at a given address. Subsequent burst mode prefetches are initiated or continued by the prefetch controller whenever the instruction queue contains only a few instruction words. Burst prefetching begins as soon as the bus is free of operand accesses previously requested by the microsequencer. Additional state information permits the controller to inhibit or terminate prefetch bursts when a change in instruction flow is executing thereby improving the change of flow performance.

EXAMPLE

Change in flow instructions are bcc, jsr, rts and jmp instructions.

In a typical program, 10 to 25 percent of the instructions cause a change of flow. Each time a change occurs, the instruction queue and instruction pipeline must be flushed and refilled from the new instruction stream. The prefetch controller and microsequencer optimize change of flow instructions by detecting the change of flow early, then terminating prefetches in progress or inhibiting unnecessary prefetches from occurring. For non-change of flow instructions the prefetch controller will schedule prefetches when adequate room is available in the queue.

2.5.2 Write-Pending Buffer

The CPU32X incorporates a single-operand write-pending buffer. The buffer permits the microsequencer to continue execution after a request for a write cycle is queued in the bus controller. The time needed for a write at the end of an instruction can overlap the beginning of the following instruction, and thus reduce overall execution time. Interlocks prevent the microsequencer from overwriting the buffer.

The write pending buffer is the same on the CPU32 as on the CPU32X.

2.5.3 Microbus Controller

The microbus controller performs bus cycles issued by the microsequencer. Operand accesses always have priority over instruction prefetches. Word and byte operands

are accessed in a single CPU-initiated bus cycle, although the external bus interface will be required to initiate a second cycle when a word operand is sent to a byte-sized external port. Long operands are accessed in two bus cycles, most significant word first.

2.6 Execution Unit

The execution unit contains all of the logic for carrying out instruction execution. This logic includes the user visible registers (A0-A7, D0-D7, SP, PC, CCR, etc.), the arithmetic logic unit, the shifter and other functional units required for executing all instructions. In addition the execution unit has access to the pipe for extension words and to the data bus buffers for reading and writing memory. The execution unit performs its operations under the control of the microsequencer.

The execution unit is the same on the CPU32 as on the CPU32X.

2.7 Instruction Execution Examples

The following are instruction execution examples to illustrate three points. First, the overlap of current instruction execution, next instruction decode and burst instruction prefetch. Second, is the interaction between operand bus cycles and the prefetching of instructions. Third, restarting the pipe, microsequencer and instruction prefetching after a change of flow instruction.

These instruction examples represent the expected system implementation. The external burst memory is a 2,1,1,1 burst memory which results in 3,1,1,1 IMB burst accesses. Accesses to IMB based peripherals and external memory are assumed to be two clock accesses. Accesses to the FASRAM are one clock accesses.

2.7.1 Execution Overlap

Table 2-1 is the instruction stream I0, I1 and I2. I0, I1 and I2 are one clock instructions with decode, prefetch and execute operations overlapping. This figure begins with the instruction queue containing instructions I4 and I5, a prefetch is initiated for I6 and the pipe is full with instruction I0 executing.

Table 2-1 Execution Overlap

Clock	1	2	3	4
IMB bus controller	Burst prefetch I6,....			I7
FASRAM	—			
Instruction queue	I4,I5	I5	—	
IR	I3	I4	I5	I6
IRA-decode	I2	I3	I4	I5
IRB	I1	I2	I3	I4
IRC-execute	I0	I1	I2	I3

2.8 Operand Accesses

There are two types of operand accesses (data). An operand access to the FASRAM and an operand access not to the FASRAM. All operand accesses, except IACK cycles, go to the FASRAM. IACK cycles are always run on the IMB.) If the operand is in the FASRAM it is returned in one clock, the instruction continues and burst prefetching continues. If the operand is not in the FASRAM the bus controller terminates the burst prefetch in progress and then runs the operand access cycle on the IMB. Once the operand access cycle completes on the IMB burst instruction prefetching is restarted and the instruction continues. An example of a move instruction accessing the FASRAM is shown in **Table 2-2**. An example of a move instruction to an operand not in the FASRAM is shown in **Table 2-3**. The instruction execution time for the move from the FASRAM is three clocks. The best case instruction execution time for the move from two-clock non-FASRAM memory is five clocks. The worst case instruction execution time for the move from two clock non-FASRAM memory is six clocks. For comparison this same move from two clock memory on the CPU32 requires six clocks.

2.8.1 Move (A0),D0 A0 = FASRAM

Table 2-2 is an example of an access to the FASRAM which occurs in parallel to instruction prefetch. Register A0 points to an address which resides in the FASRAM. In this example instructions I1 through I5 follow the move. The pipe is full and the instruction queue is as shown.

Table 2-2 Move (A0),D0 A0 = FASRAM

Clock	1	2	3	4
IMB bus controller	Burst prefetch I6,...			I7
FASRAM		Read (A0)		
Instruction queue	I4,I5	I4,I5	I4,I5	I5,I6
IR	I3	I3	I3	I4
IRA-decode	I2	I2	I2	I3
IRB	I1	I1	I1	I2
IRC-execute	move (A0),D0			I1

2.9 Move (A0),D0 A0! = FASRAM

Table 2-3 is an example of a move (A0),D0 and A0 points outside the FASRAM. In this example instructions I1 through I5 follow the move. The pipe is full, burst prefetch is in progress and the instruction queue is full as shown. I1, I2 and I3 are one clock instructions. This example is the worst case delay. If the IMB is idle, no burst in progress, the move will execute in five clocks instead of six.

Table 2-3 Move (A0),D0 A0! = FASRAM

Clock	1	2	3	4	5	6	7	8	9
IMB Bus Controller	I4	I5	I6	Read (A0)		Burst Prefetch I7,...			I8
FASRAM		Check (A0)							
Instruction Queue		I4	I4,I5	I4-6			I5,I6	I6	I7
IR	I3						I4	I5	I6
IRA-Decode	I2						I3	I4	I5
IRB	I1						I2	I3	I4
IRC-Execute	move (A0),D0						I1	I2	I3

2.10 Burst Start-up

Table 2-4 is an example of a JMP to instruction I0. The pipe and instruction queue is flushed. The jump instruction is waiting on the pipe to be refilled to complete.

Table 2-4 JMP to I0

Clock	1	2	3	4	5	6	7
IMB bus controller		Burst prefetch I0...			I1	I2	I3
FASRAM							
Instruction queue							
IR							
IRA-decode					I0	I1	I2
IRB						I0	I1
IRC-execute	JMP to I0						I0

2.11 Exceptions

Interrupt, breakpoint and bus error exceptions function the same on the CPU32X as on the CPU32. Retry and relinquish and retry (RRT) are not supported. The late bus termination is also not supported. Please refer to the Section 6 of the ***CPU32 User's Manual*, CPU32RM/AD** for information on how the CPU32X interrupt and bus error exceptions function.

2.11.1 Standard Bus Cycle Exceptions

Standard (non-burst) IACK breakpoint and bus error exception bus cycles operate the same on the CPU32X as on the CPU32. In addition interrupts and IMB peripheral breakpoints operate the same as on the CPU32. Retry, RRT and late bus error are supported by the CPU32 but are NOT supported on the CPU32X.

2.11.2 Burst Bus Cycle Exceptions

Breakpoint and bus error exception bus cycles are extended as follows to support the burst bus cycles.

2.11.2.1 Burst Breakpoint

Burst instruction breakpoints do not terminate the burst bus cycle. Individual words in a burst cycle can be breakpointed. The instruction breakpoint is taken when the word breakpointed is accessed in the pipe. If the pipe and queue are flushed by a change of flow the breakpoint will not be taken.

2.11.2.2 Burst Bus Error

Bus error terminates the burst bus cycle in progress. The bus error tags the last word transferred during the burst in the instruction queue with bus error. When the word with the bus error tag reaches the pipe the bus error exception is taken. Until the bus error exception is taken more burst prefetches and operand accesses can occur. If the instruction bus error is flushed from the pipe and queue by a branch instruction or other change of flow the bus error will not be taken.

Late bus errors on burst bus cycles are not supported.

2.12 Interrupt Response Time

The minimum interrupt response time of the CPU32X is improved over the CPU32. The CPU32 requires a minimum of thirty clocks to begin executing the first instruction of the interrupt service routine and the CPU32X will require a minimum of eighteen clocks or twelve clocks less. **Table 2-5** indicates the minimum number of clocks required and the operations performed for the CPU32 and the CPU32X to begin the interrupt service routine. This example assumes that the stack **and vector table** are in the FASRAM, external memory accesses are two cycle accesses and burst memory is a **1,1,1,1 external** burst memory, which is a **2,1,1,1 internal** burst memory. Additional clocks for slower external burst devices and non-FASRAM vector table accesses will increase the interrupt response time.

Table 2-5 Interrupt Response

Clock	CPU32X	CPU32
	Last instruction	Last instruction
1		
2	IACK cycle	
3		IACK cycle
4		
5		
6		
7		
8	Stack vector	
9		
10	Stack SR	
11	Stack PCH	Stack vector
12	Stack PCL	
13	**Read vector high**	
14	**Read vector low**	
15	Begin burst	Stack SR
16	I0	
17	I1	Stack PCH
18	I2	
19	**Begin Interrupt Routine**	Stack PCL
20		
21		Read vector high
22		
23		Read vector low
24		
25		Fetch first instruction
26		
27		Fetch second instruction
28		
29		Fetch third instruction
30		
		Begin interrupt routine

2.13 Power

The power dissipation of the greater-than-two-times faster CPU32X is substantially less than two times the CPU32 power dissipation. This power reduction is from improving the power dissipation in the CPU32 core, by stopping all unnecessary clocks and by design for power in the new burst mode, fast bus and instruction queue logic. In addition the operation of the stop instruction will be improved to reduce power when the CPU32X is stopped.

2.14 Debugging Support

The CPU32X provides features that facilitate applications development similar to the CPU32 debugging features. These features include background debug mode, breakpoint, and pipe tracking.

2.14.1 Background Debug Mode

This feature remains the same on CPU32X as on CPU32.

2.14.2 Breakpoint

This feature is enhanced from the CPU32 to provide burst breakpoints.

2.14.3 Pipe Tracking

CPU32X uses three pins. IPIPE[2:0] provides five types of information for the bus analyzer to track the pipe. $\overline{\text{IPIPE[0]}}$ and $\overline{\text{IPIPE[1]}}$ both carry active low, time multiplexed signals, and provide four types of information, which are instruction-start, opcode-advance, instruction-fetch, and pipe-flush. IPIPE[2] indicates the number of instructions in the pipe, and helps the bus analyzer synchronize with the pipe.

2.14.3.1 Instruction Queue and Pipeline

In addition to the original three-stage instruction pipeline, CPU32X implements a six-word instruction queue. The internal instruction queue and pipeline can be modeled as a nine-stage FIFO and a fetch pointer. **Table 2-6** describes the FIFO.

IR contains the currently executing instruction. IR0 contains instruction extension words. When a new opcode or extension word is used, the cpu will shift the data in the instruction registers upward, and discard the old data in IR or IR0.

The fetch pointer contains the address of the empty instruction register to be filled by the new incoming instruction word. The fetch pointer may point to IR0~IR7, but it will never directly point to IR.

Table 2-6 Nine-Stage Instruction Registers

Address	Instruction Register	Data	
	IR (IRC)	Current instruction	
000	IR0(IRB)	Extension word	
001	IR1(IRA)	New opcode	
010	IR2	New opcode	
011	IR3	New opcode	
100	IR4	Empty	← Fetch Pointer
101	IR5	Empty	
110	IR6	Empty	
111	IR7	Empty	

2.14.4 PIPE Tracking Operations

2.14.4.1 $\overline{\text{IPIPE[0]}}$

The $\overline{\text{IPIPE[0]}}$ pin provides the instruction-start and opcode-advance signals. The start signal is derived from sampling $\overline{\text{IPIPE[0]}}$ at the falling edge of the system clock, and the advance signal is derived from sampling $\overline{\text{IPIPE[0]}}$ at the rising edge of the system clock. They are both active low signals.

Instruction-start and opcode-advance both indicate the use of the new instruction word in the pipeline. Instruction-start means the start of a new instruction. The old opcode in IR is replaced by the new opcode in IR0, and all of the data in the pipeline is shifted upward, (i.e., IR0->IR, IR1->IR0, ... and IR7->IR6). The fetch pointer is decremented by one.

Opcode-advance is different from instruction-start. It indicates the use of extension word from IR0. The data in IR0 is replaced by IR1 and all of the data is shifted upward to IR0, (i.e., IR1->IR0, IR2->IR1,..., and IR7->IR6). The fetch pointer is also decremented by one.

2.14.4.2 $\overline{\text{IPIPE[1]}}$

The $\overline{\text{IPIPE[1]}}$ pin provides pipe-flush and instruction-fetch information. The flush signal is derived from sampling $\overline{\text{IPIPE[1]}}$ at the rising edge of system clock, and the fetch signal is derived from sampling $\overline{\text{IPIPE[1]}}$ at the falling edge of system clock. The fetch and flush signals are both active low.

Pipe-flush indicates when a change of flow occurs and all of the instructions in the pipeline are flushed. After pipe-flush is asserted, the pipeline is empty, and the fetch pointer is zero.

Instruction-fetch indicates that the data from the current IMB cycle is to be routed to the instruction register pointed to by the fetch pointer. After the data is copied to the instruction register, the fetch pointer is incremented by one. The fetch pointer now points to the next empty instruction register.

The instruction-fetch signal is asserted at IMB B3 state, and repeats asserted during wait states until B4.

Figure 2-3 describes the timing of $\overline{\text{IPIPE[1:0]}}$ pins and the fetch pointer.

Figure 2-3 through **Figure 2-8** describe the timing of the IPIPE[1] in different burst fetch cycle. **Figure 2-5** shows minimum logic required to demultiplex $\overline{\text{IPIPE[0]}}$ and $\overline{\text{IPIPE[1]}}$.

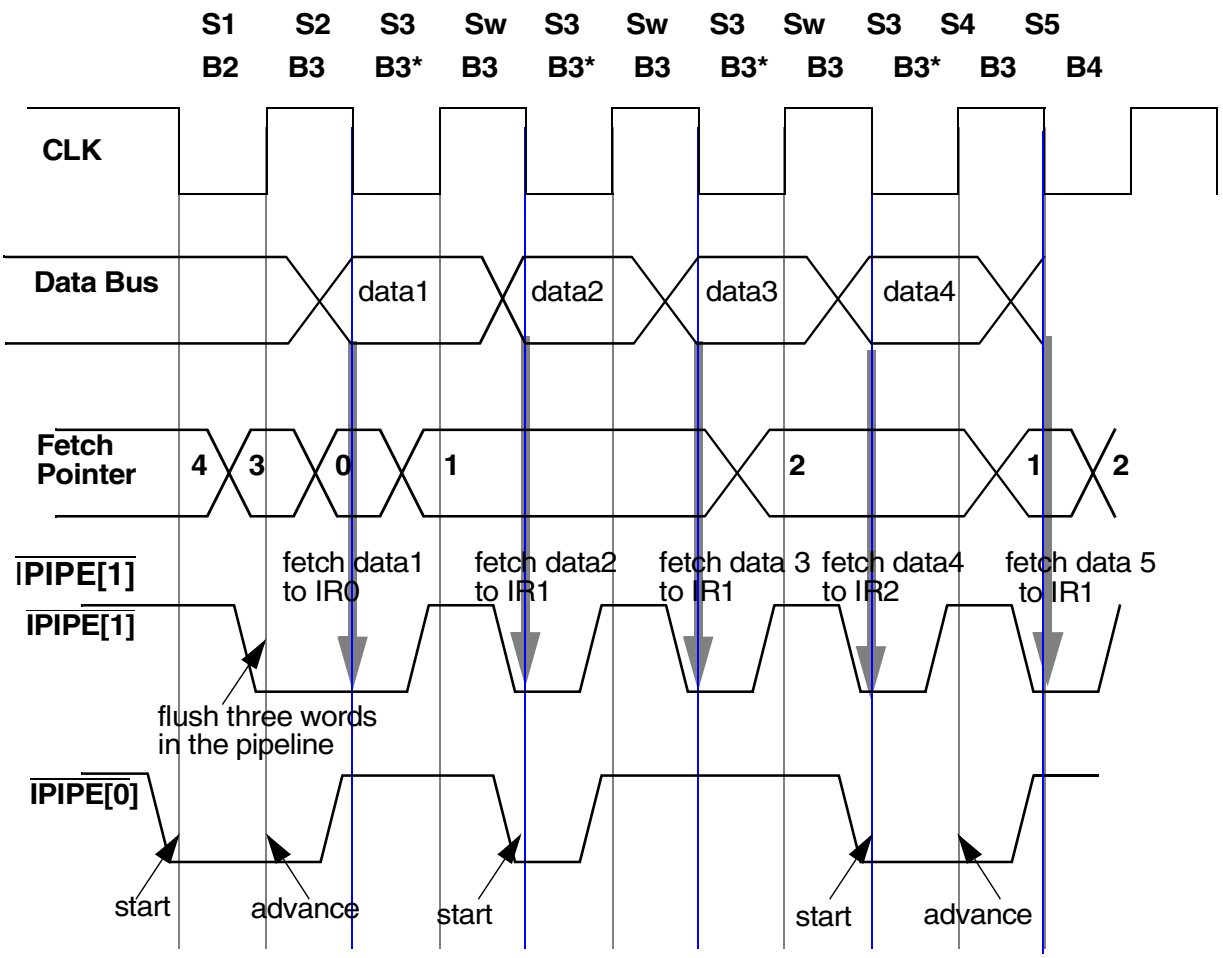


Figure 2-3 $\overline{\text{IPIPE[1:0]}}$ Timing Diagram

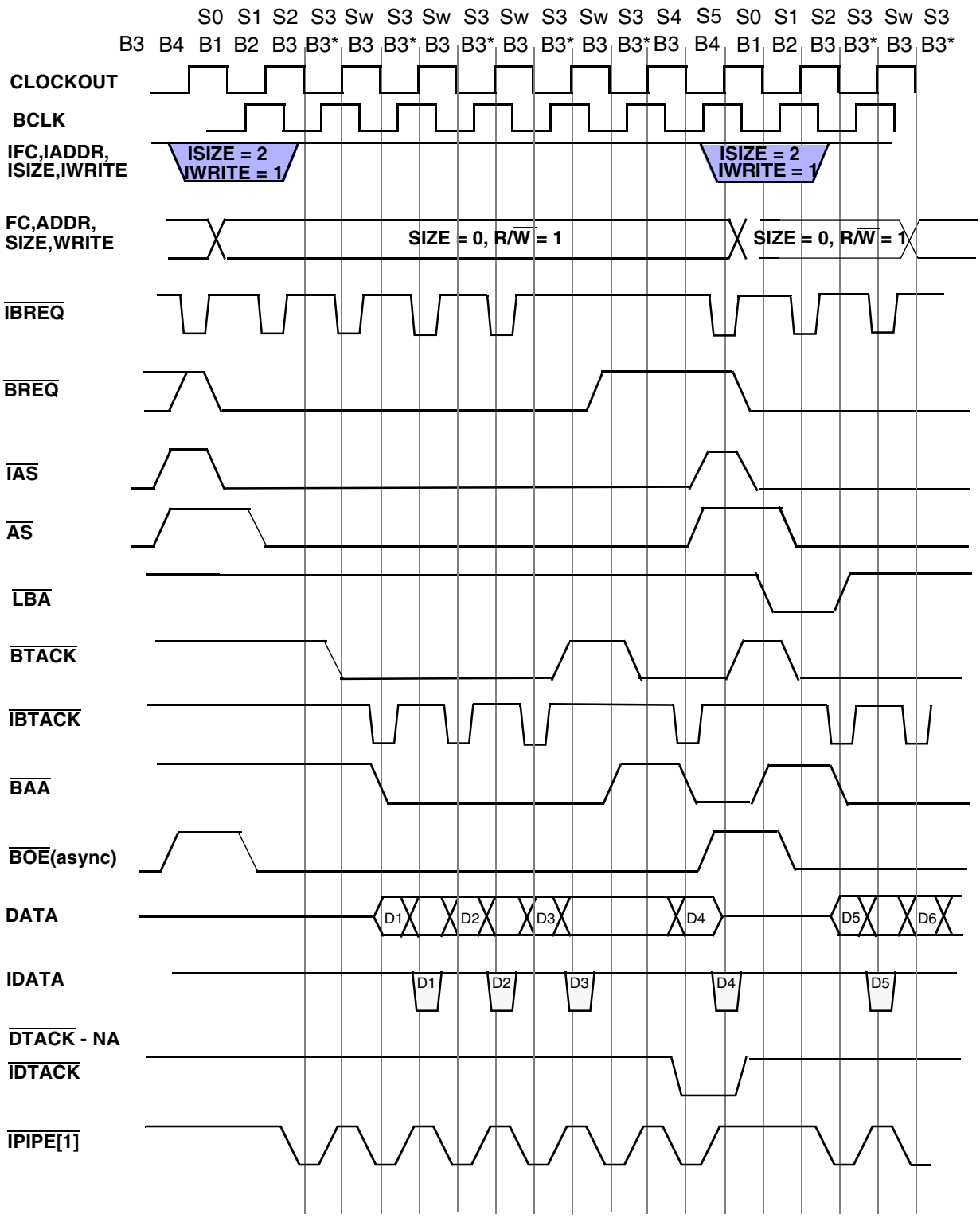
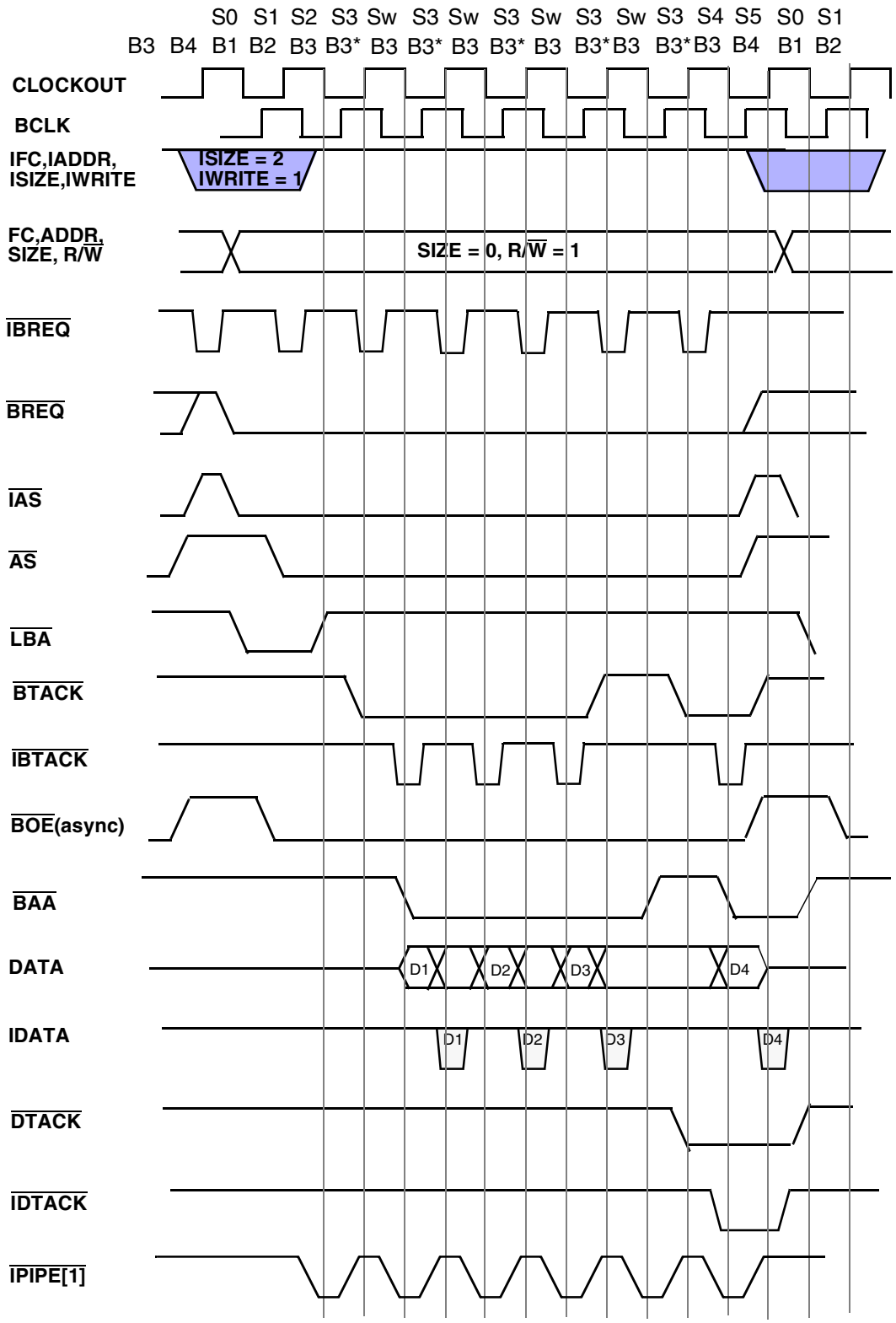
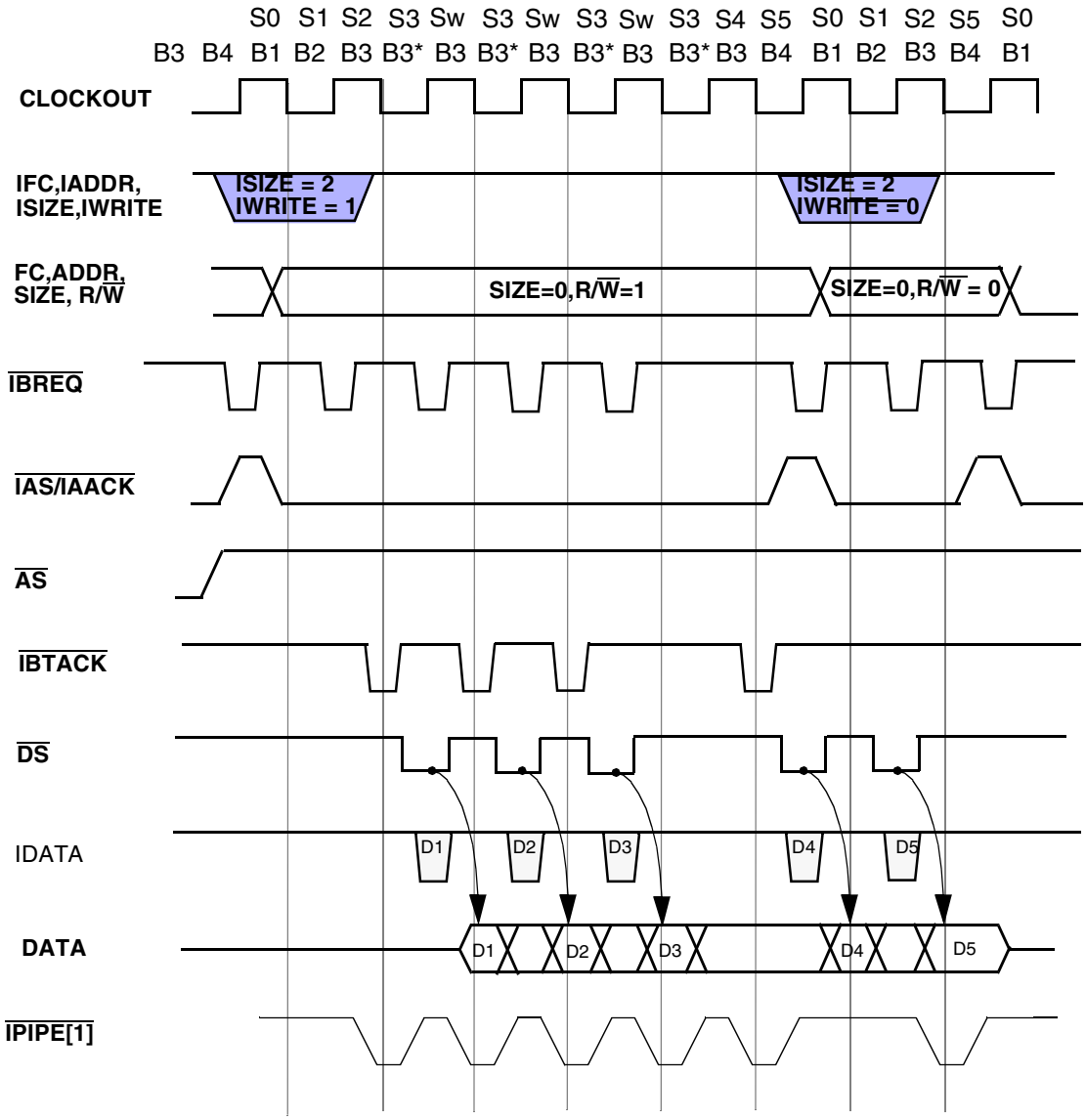


Figure 2-4 $\overline{\text{IPIPE[1]}}$ Timing Diagram In a Burst Fetch Cycle (Part 1)



NOTE: Slave terminates cycle by asserting DTACK.
IPIPE[1] indicates the data, D1~D4, enters pipeline.

IPIPE[1:0] Timing Diagram in a Burst Fetch Cycle (Part 2)



IPIPE[1:0] Timing Diagram in a Burst Fetch Cycle (Part 3)

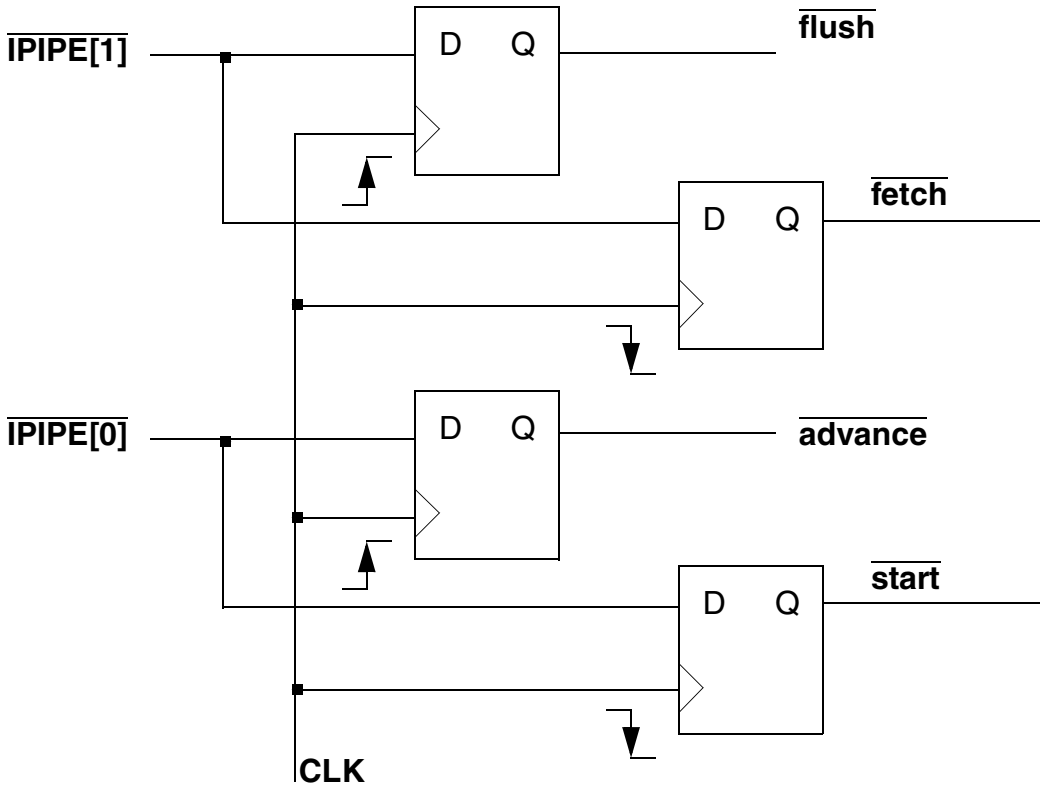


Figure 2-5 IPIPE[1:0] DEMUX Logic

2.14.4.3 IPIPE[2]

In the CPU32, the only way to synchronize the bus analyzer with the internal pipe at the first start is through a pipe_flush signal. Once the bus analyzer receives the signal, it knows the pipeline is empty and then starts the pipe tracking. However, since the pipe_flush signal depends on the change-of-flow signal from the user's program, there is a chance that the bus analyzer has to wait for a long time before the pipe_flush is generated and can not service the user at all.

In the CPU32X, IPIPE[2] pin provides the pipe synchronization function to help the bus analyzer synchronize with the internal pipe without depending on the pipe_flush signal. A series of bits (**see Figure 2-3**) are sent out by IPIPE[2] to indicate the number of instruction words in the pipeline. Once the bus analyzer gets the information, it can assign the fetch pointer to the correct address and start tracking the pipe from that time point.

The serial data bits start from one start bit ("0"), followed by three bits, indicating the number of instruction words in the pipeline, and then followed by three or more stop bits ("111"). The start bit always comes with a fetch or a flush signal, called fetch or flush. The IPIPE[2] data indicates the number of instructions in the pipeline after the fetch or flush occur. When it comes with flush, the data from the next fetch cycle must

enter IR0. When it comes with fetch, the instruction number derived from IPIPE[2] is assigned as the fetch pointer for the fetch cycle following fetch.

If there are start or advance signals detected between fetch and the next fetch, the fetch pointer has to be adjusted by subtracting the number of starts and advances to acquire the correct fetch pointer.

In the example of **Figure 2-3,** the fetch pointer is "011" after fetch1 occurs. However, a start occurs simultaneously with the following fetch cycle, so the data2 from the following fetch enters IR2 again. The fetch pointer is "101" after fetch2 occurs and therefore, data7 from the following fetch enters IR5.

Figure 2-7 lists the detailed timing for IPIPE[2:0] and shows an example of how to use IPIPE[2] to synchronize the pipe tracking function in tabular form.

In **Figure 2-7**, the first IPIPE[2] start bit is detected at line one, which comes with a flush signal; therefore, the data of the following fetch cycle enters IR0. The second IPIPE[2] start bit is detected at line 17, and the IPIPE[2] data indicates there are three instruction words in the pipeline before the next fetch (at line 19) occurs. The fetch pointer is set to '3' at line 18. However, there is another "instruction-start" detected at line19, so the fetch pointer for the fetch cycle at line19 is adjusted to two by subtracting '1'.

The third IPIPE[2] start bit is detected at line 33, which comes with a fetch cycle at line 33. The IPIPE[2] data indicates the following incoming data should enter IR0. In this case, it means the pipeline is full and if another fetch cycle occurs before any "instruction-start" or "instruction-advance", the pipe will overflow, which is impossible to happen. At line 34, the fetch pointer is assigned to '0' according to the data from IPIPE[2]. But before the next fetch cycle comes, an "instruction-start" and "instruction-advance" are detected, and the fetch pointer should be adjusted by adding '8' and subtracting '2' for the fetch cycle at line 39.

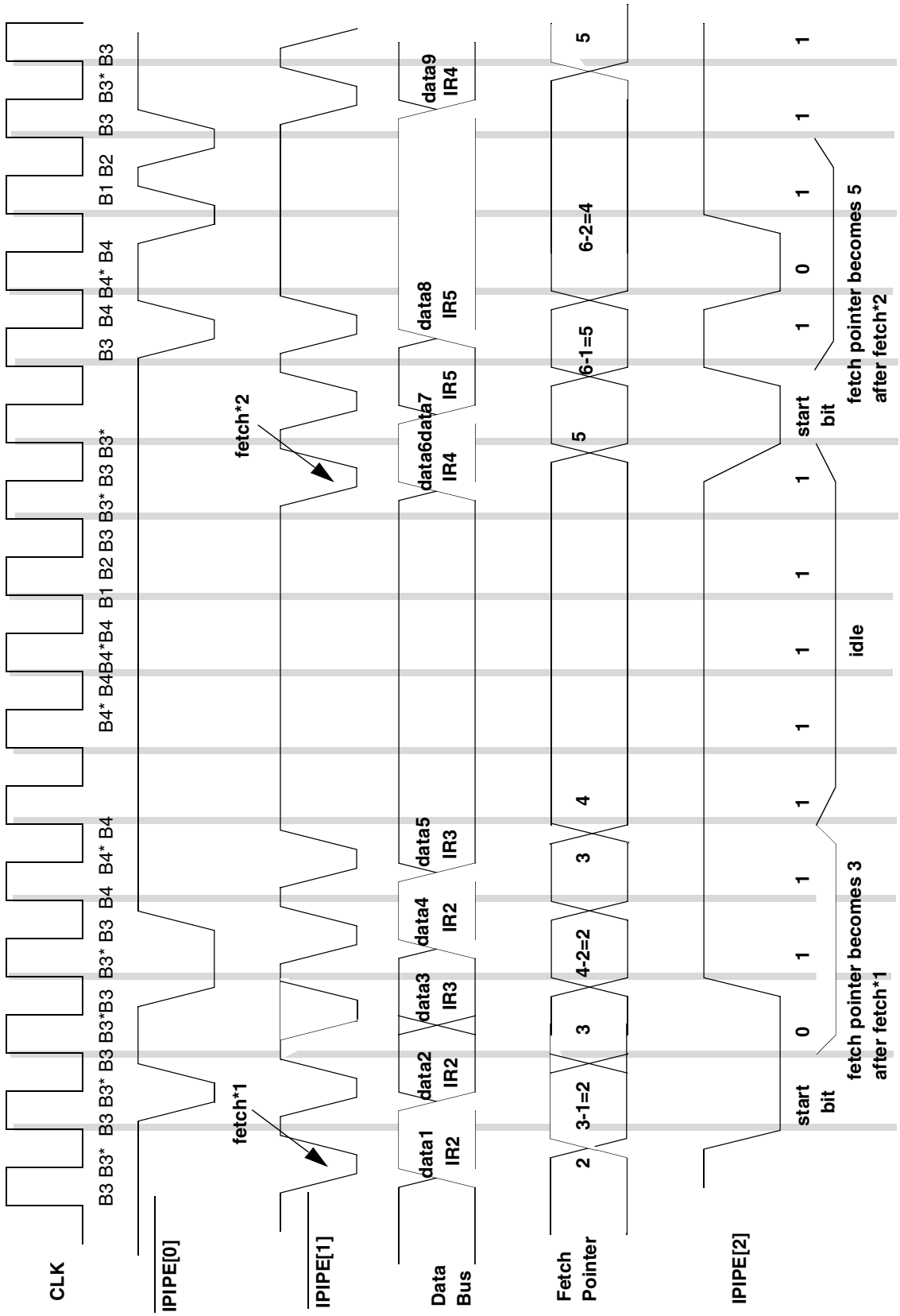


Figure 2-6 $\overline{\text{IPIPE[2]}}$ Timing Diagram

Table 2-7 How to Use $\overline{\text{IPIPE[2]}}$ to Synchronize Pipe Tracking

line	clkout	ipipe2	flush	advance	fetch	start	latch data	data ->IR?	fetch pointer
1	^	0	1	1	x	1	xxxx		
2		0	1	1	x	1	xxxx		0
3	^	0	0	1	x	1	xxxx		0
4		0	0	1	0	1	2a7c		0
5	^	0	1	1	0(+)	1	2a7c	IR0	0
6		0	1	1	0	1	0020		1
7	^	0	1	1	0(+)	1	0020	IR1	1
8		0	1	1	0	1	4a00		2
9	^	1	1	1	0(+)	1	4a00	IR2	2
10		1	1	1	0	0	4e68		3
11	^	1	1	0(-)	0(+)	0(-)	4e68	IR1	1
12		1	1	0	0	1	4e60		2
13	^	1	1	0(-)	0(+)	1	4e60	IR1	1
14		1	1	0	0	1	4e7a		2
15	^	1	1	1	0(+)	1	4e7a	IR2	2
16		1	1	1	0	0	6801		3
17	^	0	1	1	0(+)	0(-)	6801	IR2	2
18		0	1	1	0	0	48d5		3
19	^	0	1	1	0(+)	0(-)	48d5	IR2	2
20		0	1	1	0	0	5555		3
21	^	1	1	0(-)	0(+)	0(-)	5555	IR1	1
22		1	1	0	0	1	0f8e		2
23	^	1	1	1	0(+)	1	0f8e	IR2	2
24		1	1	1	0	1	0000		3
25	^	1	1	1	0(+)	1	0000	IR3	3
26		1	1	1	0	1	700a		4
27	^	1	1	1	0(+)	1	700a	IR4	4
28		1	1	1	0	1	0e15		5
29	^	1	1	1	0(+)	1	0e15	IR5	5
30		1	1	1	0	1	1800		6
31	^	1	1	1	0(+)	1	1800	IR6	6
32		1	1	1	0	1	c7c2		7
33	^	0	1	1	0(+)	1	c7c2	IR7	7
34		0	1	1	1	0			0
35	^	0	1	0(-)	1	0(-)			0-2+8
36		0	1	0	1	1			6
37	^	0	1	1	1	1			6
38		0	1	1	0	1	xxxx		6
39	^	0	1	1	0(+)	1	xxxx	IR6	6
40		0	1	1	1	1			7

2.14.4.4 Loop Mode

$\overline{\text{IPIPE[1]}}$ and IPIPE[2] continue to work normally during loop mode. However, unlike CPU32, CPU32X does not discard any opcode since the opcodes are being reused during loop mode. The start and advance signals will not be asserted during loop mode. From the user's point of view, loop mode is like running a long instruction. **Figure 2-8** is an example of loop mode.

Assembly Code

```
203c  0000 0000          move.l #$0,d0
207c  0000 1000          move.l #$1000,a0
d441                     add d1,d2
30c0                lp   move d0,(a0)+
54ca  fffc               dbcc d2,lp
d642                     add d2,d3
d84b                     add d3,d4
                          ⋮
```

sync	fetch		advance						fetch	operand	
	flush		start		IR	IR0	IR1	IR2	opcode	addr.	data
1	1	1	1	1	d441	30c0	54ca	fffc			
1	1	1	0	1	30c0	54ca	fffc	d642			
1	1	1	1	1	30c0	54ca	fffc	d642		1000	0000
1	1	1	0	1	54ca	fffc	d642	d84b			
0	1	1	1	0	54ca	d642	d84b				
0	0*	1	1	1	xxxx	xxxx	xxxx	xxxx			
0	1	1	1	1	xxxx	xxxx	xxxx	xxxx			
0	1	0	1	1	xxxx	30c0	xxxx	xxxx	30c0		
1	1	0	0	1	30c0	54ca	xxxx	xxxx	54ca		
1	1	0	1	1	30c0	54ca	fffc	xxxx	fffc	1002	0000
1	1	0	0	1	54ca	fffc	d642	xxxx	d642		
1	1	0	1	1	54ca	fffc	d642	d84b	d84b	1004	0000
1	1	0	1	1	54ca	fffc	d642	d84b			
1	1	0	1	1	54ca	fffc	d642	d84b	⋮	1006	0000
1	1	1	1	1	54ca	fffc	d642	d84b			
1	1	1	1	1	54ca	fffc	d642	d84b		1008	0000
1	1	1	1	1	54ca	fffc	d642	d84b			
1	1	1	1	1	54ca	fffc	d642	d84b		100a	0000
1	1	1	1	1	54ca	fffc	d642	d84b			
1	1	1	1	0	54ca	d642	d84b				
1	1	1	0	1	d642	d84b					

Loop Mode

Figure 2-7 Pipe Tracking Operation In Loop Mode

2.14.5 Pipe Tracking Flowchart

Figure 2-9 is the pipe tracking flowchart. The bus analyzer should follow it to track the pipeline.

Master terminates cycle by negating IBREQ. IPIPE[1] indicates the data, D1-D6, enters pipeline.

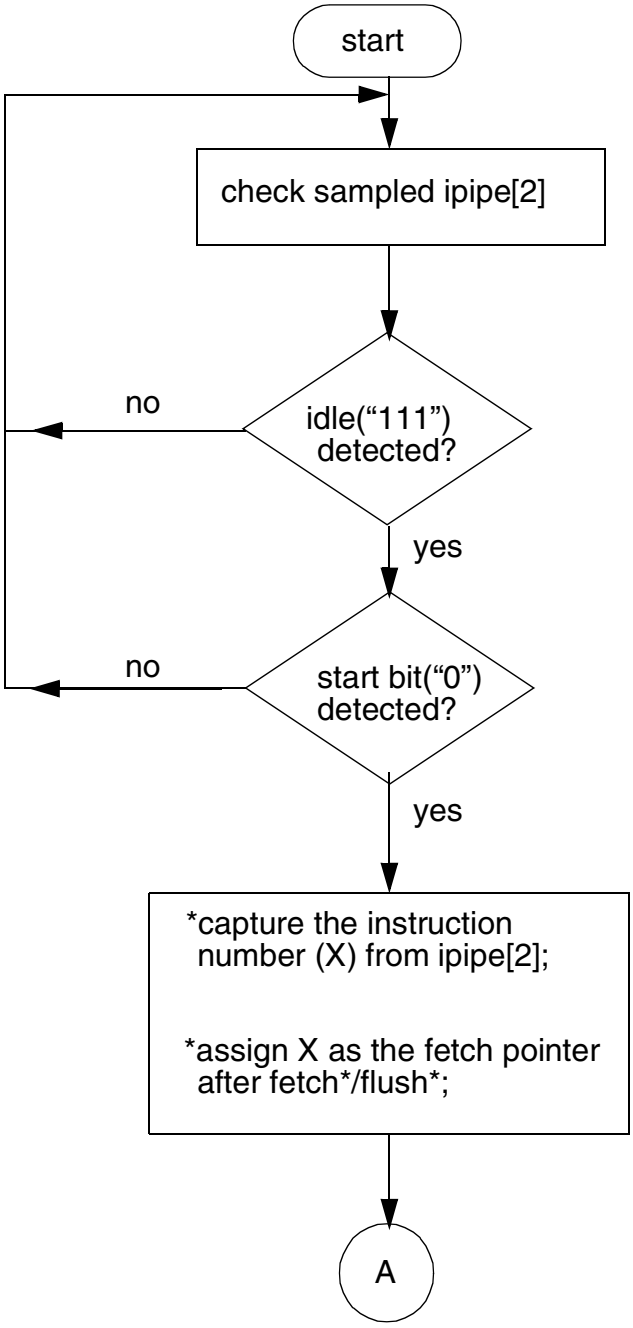


Figure 2-8 Pipe Tracking Flowchart (Part 1)

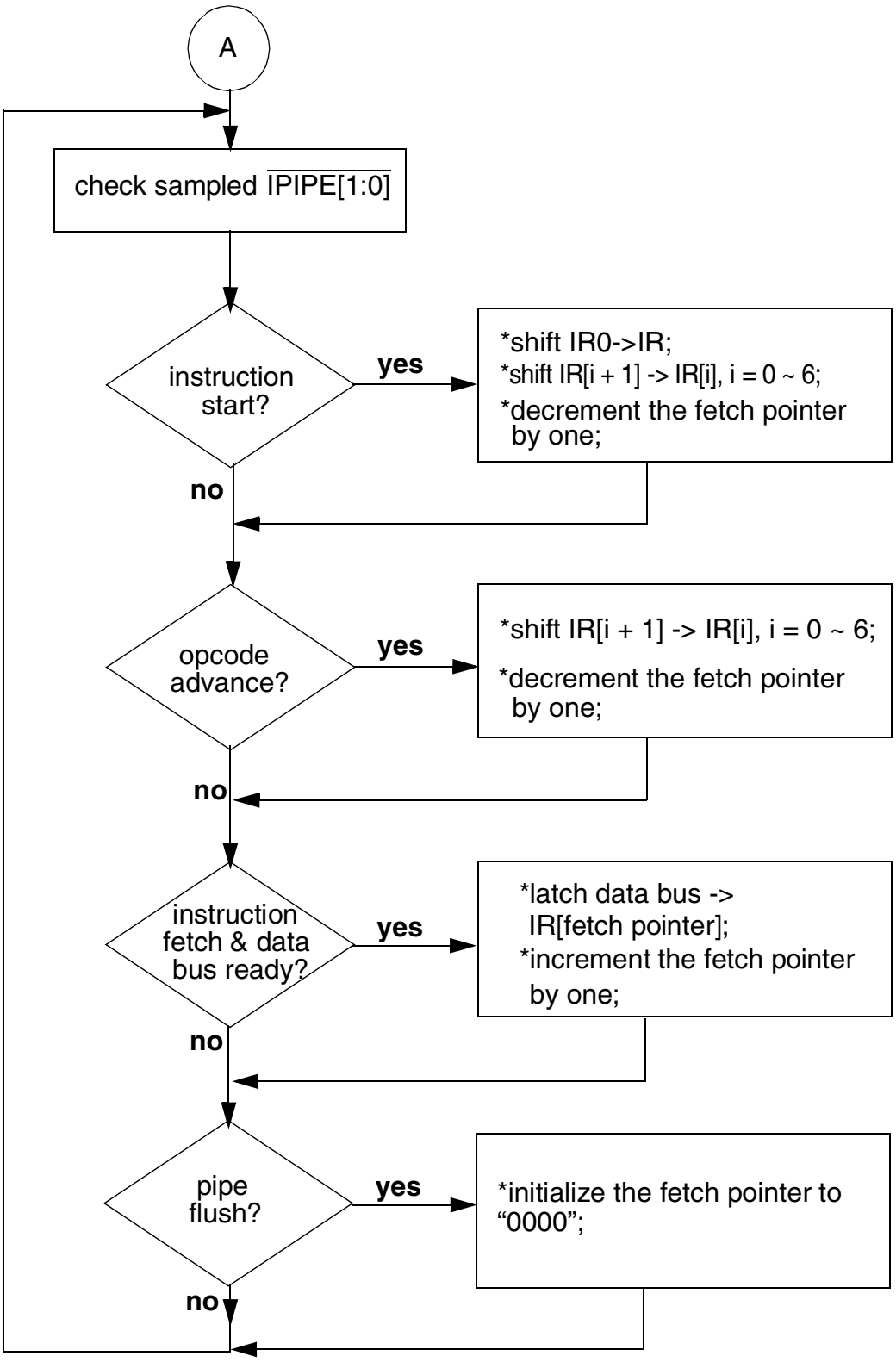


Figure 2-9 Pipe Tracking Flowchart (Part 2)

2.15 IMB Interface and External Pins

All of the IMB pins currently supported by the CPU32 will be supported by the CPU32X. The IMB burst protocol pins, BTACK and BREQ are added to the IMB interface. One pin will be added, in addition to the IPIPE and IFETCH pins, to support pipe and instruction queue tracking.

2.16 Basic Electrical Specifications

Table 2-8 Electrical Specifications

Operating Range	Temperature Range in C	Min/Max VDD in V	Max Fsys in MHz	Max Current in mA
Automotive	-40 to 125	4.9 to 5.1	21.75	90
Commercial	-40 to 110	4.75 to 5.25	25	105
Low voltage commercial	-40 to 85	2.7 to 3.3	8	20

2.17 Test Features

The test features of the CPU32X will include at least the test features of the CPU32. Any additional test features are to be determined.