CS232 Final Exam
May 5, 2006

Name: _____

- This exam has 10 exciting pages, including this handsome cover.
- There are five breathtaking questions, worth a total of 100 points.
- The final three pages are for your reference and can be detached from the exam for your convenience. They include: 1) a summary of the MIPS instruction set, 2) a datapath diagram of a pipelined MIPS processor, and 3) a reference on ASCII coding.
- No other written references or calculators are allowed.
- Write clearly and show your work. State any assumptions you make.
- You have ~2 fun-filled hours. Budget your time, and good luck!

Question	Maximum	Your Score
1	25	
2	15	
3	20	
4	15	
5	25	
Total	100	

Question 1: MIPS programming (25 points)

Below is a function that reads a file and counts the number of times the string "cs" exists in the file. It does this with a simple two-state finite-state machine, where the first state (state == 0) is the state where the previous character was **not** 'c' and the second state (state == 1) is when the previous character was 'c.' The code uses the `fgetc` function (prototype shown to the right) to get the next character in the FILE; `fgetc` returns `EOF` (end of file) when there are no more characters in the file. Write MIPS code for the `count_sequence` function (your code should call `fgetc`, but not implement it). Be sure to observe the MIPS calling convention. Refer to the attached MIPS and ASCII references.

```
int count_sequence(FILE *fptr) {
  int count = 0, state = 0, c;

  while ((c = fgetc(fptr)) != EOF) {
     if ((state == 1) && (c == 's')) {
        count ++;
        state = 0;
     } else if (c == 'c') {
        state = 1;
     } else {
        state = 0;
     }
  }
  return count;
}
```

```
int  fgetc(FILE *);
#define   EOF   (-1)
```

```
## int count_sequence(FILE *fptr) {
count_sequence:
          sub       $sp, $sp, 16
          sw        $s0, 0($sp)
          sw        $s1, 4($sp)
          sw        $s2, 8($sp)
          sw        $ra, 12($sp)
          move      $s0, $a0

##   int count = 0, state = 0, c;
          li        $s1, 0
          li        $s2, 0

##   while ((c = fgetc(fptr)) != EOF) {
cs_loop:
          move      $a0, $s0
          jal       fgetc
          beq       $v0, -1, cs_done

##         if ((state == 1) && (c == 's')) {
##                   count ++;
##                   state = 0;

          bne       $s2, 1, cs_case2
          bne       $v0, 115, cs_case2
          li        $s2, 0
          add       $s1, $s1, 1
          j         cs_loop

##         } else if (c == 'c') {
##                   state = 1;

cs_case2:
          bne       $v0, 99, cs_case3
          li        $s2, 1
          j         cs_loop

##         } else {
##                   state = 0;
cs_case3:
          li        $s2, 0
          j         cs_loop

##         }
##   }
##   return count;

cs_done:
          move      $v0, $s1

          lw        $s0, 0($sp)
          lw        $s1, 4($sp)
          lw        $s2, 8($sp)
          lw        $ra, 12($sp)
          add       $sp, $sp, 16
          jr        $ra
## }
```

Question 2: Processor and Memory System Performance (15 points)

Execution Time = #instructions * CPI * clock_period
CPI = Cycles Per Instruction
AMAT = hit_time + (miss_rate * miss_penalty)
Memory Stall Cycles = # memory operations * miss_rate * miss_penalty

Consider a program whose execution consists of: 50% arithmetic instructions, 20% loads, 10% stores, and 20% branches (25% not-taken, 75% taken). For half of the arithmetic and load instructions, the subsequent instruction is a dependent arithmetic or load instruction.

Part (a)

Consider the pipelined MIPS processor discussed in this class and whose pipeline is shown on page 9 of this exam. This machine predicts branches as not-taken and resolves branches in the ID stage of the pipeline, flushing any incorrect instructions on a branch misprediction. The processor includes forwarding from both the EX/MEM and MEM/WB registers to either input of the functional units. Assuming a perfect memory system (no stalls due to cache misses), compute the cycles per instruction (CPI) for this machine. (5 points)

Stalls when load followed by either arithmetic or load instructions: (20% loads * 50%) * 1 cycle = .1 CPI
Flushes due to taken branches: (20% branches * 75% taken) * 1 cycle = .15 CPI
Total CPI = 1 + stalls + flushes = 1 + .1 + .15 = 1.25 CPI

Part (b)

Consider the following realistic memory hierarchy described below: *Note: the next level of the hierarchy is accessed only after a miss is detected in the current level.*

L1: two-way set-associative, write-back, write-allocate 16KB cache with 32B blocks and 1 cycle access time.
L2: eight-way set-associative, write-back, write-allocate 1MB cache with 32B blocks and 8 cycle access time.
Memory: 200 cycle access time.

Assuming the instruction mix above, when 1000 instructions are executed, the processor encounters, on average, 50 L1 data cache misses and 3 L2 data cache misses, what is the average memory access time (AMAT) in cycles? *(*5 points)

L1_miss_rate = 50 misses / (1000 inst * (20% loads + 10% stores)) = 50/300 = 1/6
L2_miss_rate = 3 misses ./ 50 accesses = 3/50

L2_AMAT = L2_hit_time + L2_miss_rate * memory_access_time = 8 + 3/50 * 200 = 8 + 12 = 20
AMAT = L1_hit_time + L1_miss_rate * L2_AMAT = 1 + 1/6 * 20 = 1 + 10/3 = 4 1/3

Part (c)

Compute a new CPI for the processor described in Part (a) using the memory system described in Part (b). (5 points)

CPI_with_mem = CPI_w/o_mem + (AMAT - 1) = 1.25 + (4.3333 - 1) = 4.583333..

Question 3: Input/Output Systems (20 points)

Assume a hard drive that has a 6,000 rpm rotational speed, a 3ms average seek time, and that overhead is negligible. Each disk track has 64 sectors, and each sector holds 1 KB of data. The drive can read or write data as fast as it rotates. For this problem, to make the computations easier, you may assume that 1 KB = 1,000 bytes and 1 MB = 1,000 KB and 1 GB = 1,000 MB. For parts (a)-(c), compute the time to read a 1 GB file in the following three scenarios. In each case you are welcome to leave your result as an expression.

Part (a)
The file is randomly spread across disk sectors. (5 points)

1GB/1KB/sector = 1M sectors rot_period = 1/6000rpm * 60sec/m = 1/100 s = 10ms
Time to read one sector = seek_time + rot_delay + transfer_time = 3ms + . 5*10ms + (10ms/64) = 8.15625 ms
Total time = 1M * 8.15625ms = 8156 sec = 2.265 hours

Part (b)
The file is written sequentially onto tracks, but the tracks are spread randomly across the disk. (5 points)

tracks = 1M sectors / 64 sectors/track =
Time to read one track = seek_time + (1 or 0)*rot_delay + rot_period = 13ms or 18ms
Total time = 1M/64 * (13ms or 18ms) = 203 or 281 sec = 3.4 or 4.7 min

(We'll accept assumptions of either needing to wait for rotational delay or not)

Part (c)
The file is written sequentially onto sequential tracks. You can assume it takes only 1ms to seek to a neighboring track. (5 points)

tracks = 1M sectors / 64 sectors/track =
Time to read one track = seek_time + (1 or 0)*rot_delay + rot_period = 11ms or 16ms
Total time = 1M/64 * (11ms or 16ms) = 2.9 or 4.2 min

(We'll accept assumptions of either needing to wait for rotational delay or not)

Part (d)
Explain the difference between latency and throughput and give examples of when each would be an appropriate metric. (5 points)

Latency is the time to do one thing (units = time) , throughput is the rate which many things are done (units = things/time).

At a bank you would consider the time it took for one customer to be served a latency, but the rate at which the bank served the customers would be a throughput.

Question 4: Pipeline Implementation (15 points)

Part (a)

In lecture, we discussed how the pipeline handled conditional branches. These aren't the only kinds of control flow. To other kinds of important control flow are the jump-and-link (`jal`) instructions used for function calls and jump-register (`jr`) instructions used for return. In what pipeline stage would you expect `jal` and `jr` to be resolved (i.e., when can the next PC be known for sure)? (Note: They may be resolved in different pipeline stages.) Justify your answer. You can ignore dependences with other instructions in the pipeline. (5 points)

jal would likely be resolved in the ID stage because there is no condition to compute and the branch target can be computed when the instruction is decoded.

jr would likely be resolved in ID also, because at the end of the ID stage we've read the branch target out of the register file.

Part (b)

In the pipeline datapath at the end of the exam, there is no forwarding datapath present for the following dependence.

```
lw      $12, 0($11)
sw      $12, 0($13)
```

How many stall cycles are necessary to handle this hazard? Explain. (5 points)

A single cycle stall would enable forwarding from the mem/wb latch to the ALU in EX using the existing bypass datapath (through the rt_mux and imm_mux).

It would also be correct to stall the machine for two cycles, keeping the sw instruction in the ID stage so that the register write of the lw and the register read of the sw are done in the same cycle.

Part (c)

Write an equation for the hazard detection logic for the above stall condition. (5 points)

```
if (IF/ID.mem_write && ID/EX.mem_read &&
   ID/EX.rt == IF/ID.rt) {
                stall;
}
```

It is also alright to use EX/MEM and ID/EX for ID/EX and IF/ID (respectively) above. Also, accept ID/EX.rd for rt.

Question 5: Conceptual Questions (25 points)

Part (a)

A coding scheme has two valid code words (1010 and 0101) and uses a error detection/correction scheme. Below, we've shown how this ECC scheme handles a sample of errors.

1000 - Error Corrected to 1010	0110 - Error Detected (cannot correct)
1111 - Error Detected (cannot correct)	0111 - Error Corrected to 0101

Based on the above sample, determine how the following errors will be handled. State any assumptions.

1001 - Error Detected (cannot correct): Hamming distance 2 from both code words

1110 - Error Corrected to 1010: Hamming distance 1

0001 - Error Corrected to 0101: Hamming distance 1

0000 - Error Detected (cannot correct): Hamming distance 2 from both code words

This is __1__ bit correction, ___2__ bit detection scheme. *Fill in the blanks*. (5 points)

Part (b)

Consider a machine with a byte-addressable memory that uses 32b addresses for both virtual and physical addresses. Compute the size of a single-level page table if 4KB pages are used. (5 points)

Page table size = # pages * storage/page = 2^{20} * 4 bytes = 4MB

# pages = virtual address space size / page size = 2^{32} / 4KB = 2^{32} / 2^{10} = 2^{20} page
PPNsize = PA_bits - page_offset_bits = 32 - 10 = 22b
Storage per page = round up (PPN size + 1 (dirty) + 1 (valid)) = round up (22b + 1 + 1) = 32b = 4B

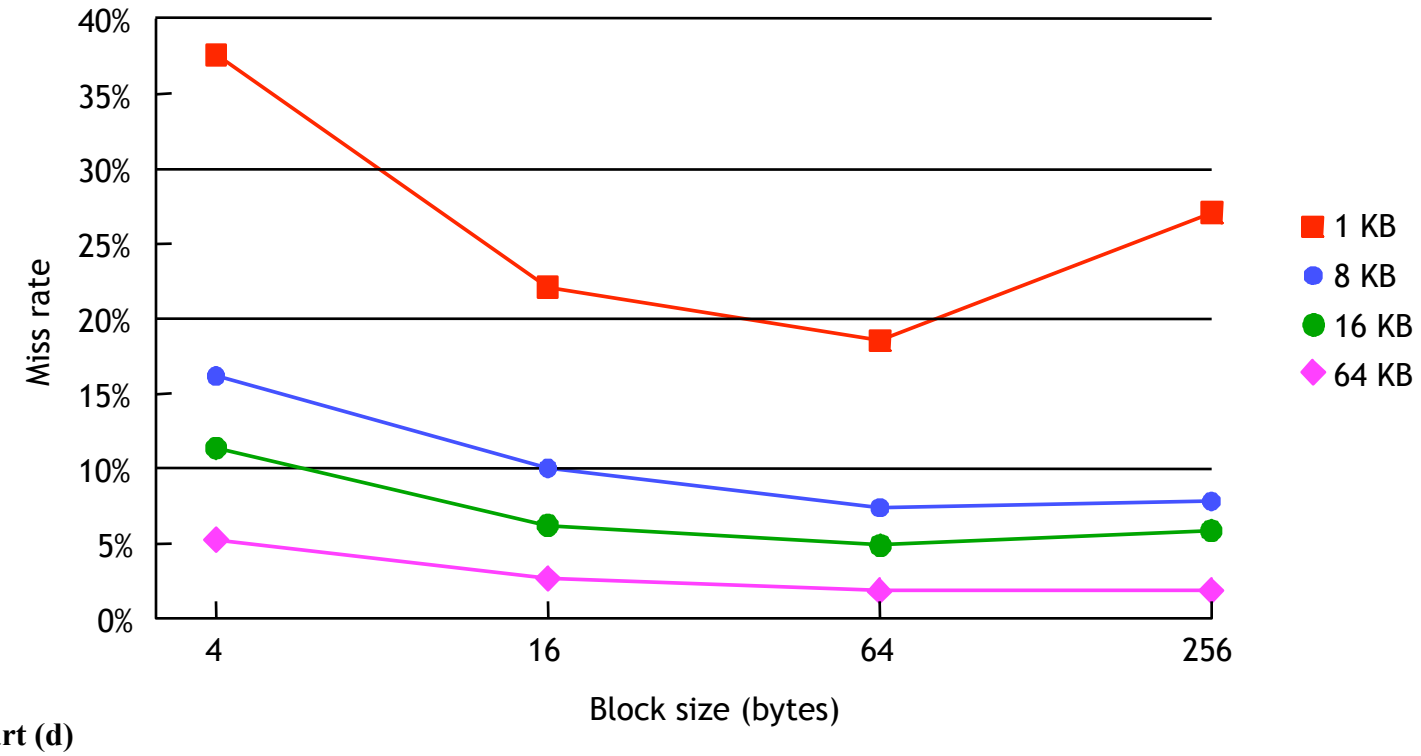
Part (c)

Describe how a multi-level page table works and explain its advantages relative to a single-level page table. (5 points)

For the multi-level page table, the virtual page number (VPN) is broken into pieces (VPN_1 … VPN_n). The first piece is used to look up in the first-level page table (pointed to by the page table base register), the entry found is a pointer to the next level page table. This pointer is used with the next piece of the VPN until the last level of the page table is found which holds the PPN.

The main advantage of using the multi-level page table is that you only need to allocate memory for those subsets of the virtual address space that are being used. Since there is spatial locality generally the parts of the virtual address space being used are close together.

Question 5: Conceptual Questions, cont.



Part (d)

The above figure shows average miss rates for caches of different sizes and block sizes, measured on a variety of programs. The miss rate for the 1KB cache initially decreases with increased block size, but then begins to increase again; explain why this occurs. (5 points)

Initially since the blocks are small, we aren't getting much benefit from spatial locality resulting in extra misses when spatial locality exists. As we increase the block size the number of blocks decreases (at 256B blocks there are only 4 blocks); eventually the number of conflict misses out weigh the benefit of additional temporal locality.

Part (e)

When parallelizing a program for fork-join parallelism (like OpenMP) what factors would prevent a two-core machine from doubling the performance of a single-core machine? Explain at least two. (5 points)

1. Any portion of the program that can not be parallelized (serial portion, Amdahl's law)
2: Overhead resulting from forking and joining
3: Overhead due to synchronization (critical sections, etc.)
4: Overhead due to trure/ false sharing in the cache

MIPS instructions

These are some of the most common MIPS instructions and pseudo-instructions, and should be all you need. However, you are free to use *any* valid MIPS instructions or pseudo-instruction in your programs.

Category	Example Instruction	Meaning
Arithmetic / Logical	add $t0, $t1, $t2	$t0 = $t1 + $t2
	sub $t0, $t1, $t2	$t0 = $t1 – $t2
	and $t0, $t1, $t2	$t0 = $t1 & $t2
	mul $t0, $t1, $t2	$t0 = $t1 x $t2
	xor $t0, $t1, $t2	$t0 = $t1 ^ $t2
	srl $t0, $t1, $t2	$t0 = $t1 << $t2
Register Setting	move $t0, $t1	$t0 = $t1
	li $t0, 100	$t0 = 100
Data Transfer	la $t0, label	$t0 = address of data at label
	lw $t0, 100($t1)	$t0 = Mem[100 + $t1] (32 bits)
	lh $t0, 100($t1)	$t0 = Mem[100 + $t1] (16 bits)
	lb $t0, 100($t1)	$t0 = Mem[100 + $t1] (8 bits)
	sw $t0, 100($t1)	Mem[100 + $t1] = $t0 (32 bits)
	sh $t0, 100($t1)	Mem[100 + $t1] = $t0 (16 bits)
	sb $t0, 100($t1)	Mem[100 + $t1] = $t0 (8 bits)
Branch	beq $t0, $t1, Label	if ($t0 = $t1) go to Label
	bne $t0, $t1, Label	if ($t0 ≠ $t1) go to Label
	bge $t0, $t1, Label	if ($t0 ≥ $t1) go to Label
	bgt $t0, $t1, Label	if ($t0 > $t1) go to Label
	ble $t0, $t1, Label	if ($t0 ≤ $t1) go to Label
	blt $t0, $t1, Label	if ($t0 < $t1) go to Label
Set	slt $t0, $t1, $t2	if ($t1 < $t2) then $t0 = 1 else $t0 = 0
	slti $t0, $t1, 100	if ($t1 < 100) then $t0 = 1 else $t0 = 0
Jump	j Label	go to Label
	jr $ra	go to address in $ra
	jal Label	$ra = PC + 4; go to Label

The second source operand of the arithmetic and branch instructions may be a constant.

Register Conventions

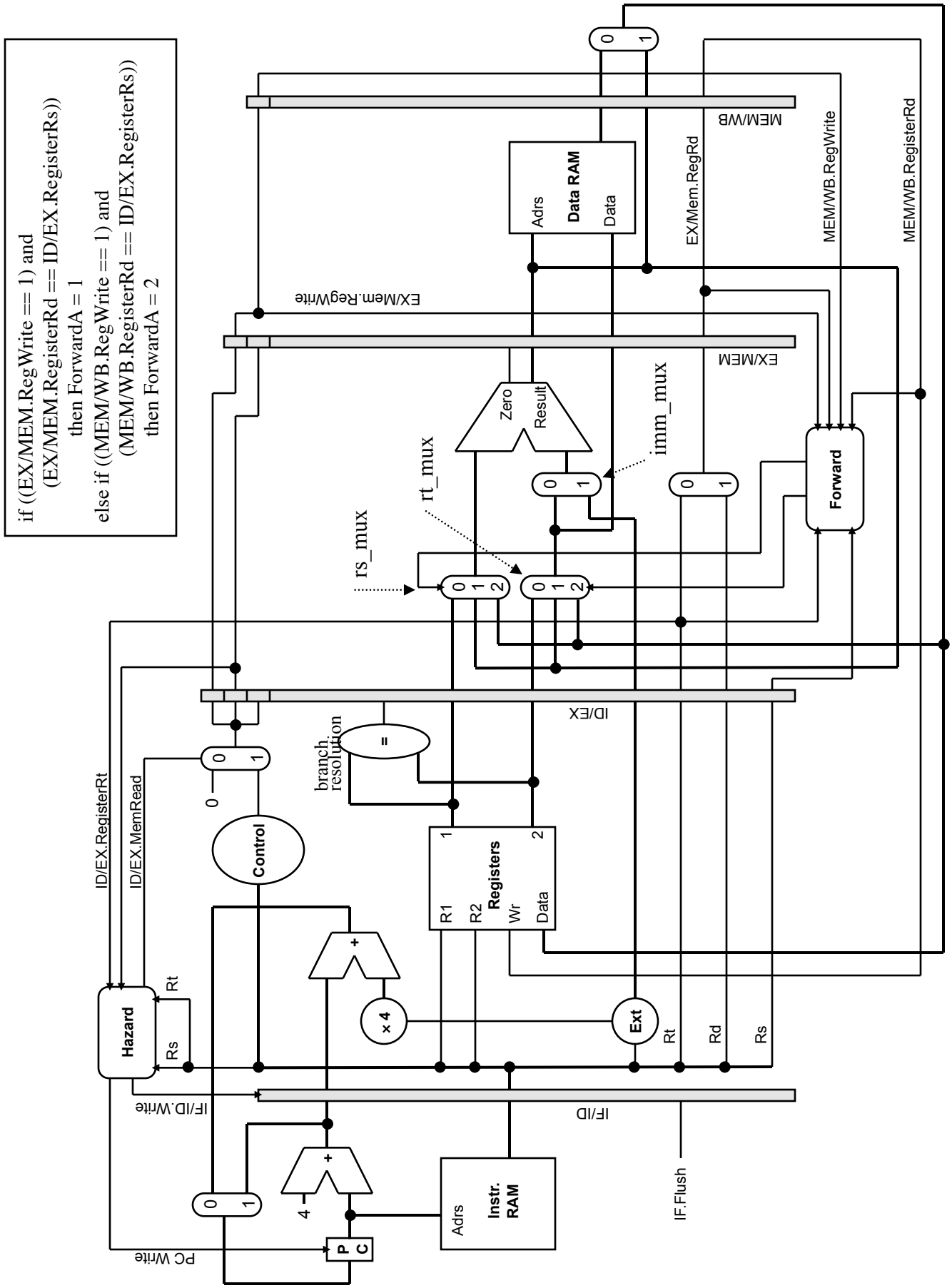
The *caller* is responsible for saving any of the following registers that it needs, before invoking a function.

$t0-$t9 $a0-$a3 $v0-$v1

The *callee* is responsible for saving and restoring any of the following registers that it uses.

$s0-$s7 $ra

Pipelined Datapath



ASCII refernence

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

ASCII stands for American Standard Code for Information Interchange. Computers can only understand numbers, so an ASCII code is the numerical representation of a character such as 'a' or '@' or an action of some sort. ASCII was developed a long time ago and now the non-printing characters are rarely used for their original purpose. Below is the ASCII character table and this includes descriptions of the first 32 non-printing characters. ASCII was actually designed for use with teletypes and so the descriptions are somewhat obscure. If someone says they want your CV however in ASCII format, all this means is they want 'plain' text with no formatting such as tabs, bold or underscoring - the raw format that any computer can understand. This is usually so they can easily import the file into their own applications without issues. Notepad.exe creates ASCII text, or in MS Word you can save a file as 'text only'