Comp Arch - HW4 Group Project

Group Members

Max Graves (mgraves4)	Aaron Lewis (alewis25)
Sam Monterola (smontero)	Charlie Gussen (cgussen)

Report

Part 4.1 (dllist_step1.S)

Input your RISC-V code here:

```
54  # This procedure clears 128 bits (16 bytes) of memory
55  # previously allocated to a dll_node
56  ########################################
57  FREE_DLLIST:
58
59  # Must clear the memory pointed to by x12 (16 bytes)
60  sw x0, 0(x12)           # Clears bytes 0-3
61  sw x0, 4(x12)           # Clears bytes 4-7
62  sw x0, 8(x12)           # Clears bytes 8-11
63  sw x0, 12(x12)          # Clears bytes 12-15
64
65  jalr x0, x1, 0          # jump and link back to main
66
67  END:
68  nop
```

Reset | Step | Run | CPU: 32 Hz

```
[line 10]: lui x5, 0x7ffff
[line 11]: addi x18, x5, 0x620
[line 12]: add x5, x0, x0
[line 16]: add x12, x18, x0
[line 17]: jal x1, CALLOC_DLLIST
```

Init Value	Register	Decimal	Hex	Binary
0	x0 (zero)	0	0x00000000	0b00000000000000000000000000000000
0	x1 (ra)	20	0x00000014	0b00000000000000000000000000010100
0	x2 (sp)	0	0x00000000	0b00000000000000000000000000000000
0	x3 (gp)	0	0x00000000	0b00000000000000000000000000000000
0	x4 (tp)	0	0x00000000	0b00000000000000000000000000000000
0	x5 (t0)	0	0x00000000	0b00000000000000000000000000000000
0	x6 (t1)	0	0x00000000	0b00000000000000000000000000000000
0	x7 (t2)	0	0x00000000	0b00000000000000000000000000000000
0	x8 (s0/fp)	0	0x00000000	0b00000000000000000000000000000000
0	x9 (s1)	0	0x00000000	0b00000000000000000000000000000000
0	x10 (a0)	0	0x00000000	0b00000000000000000000000000000000
0	x11 (a1)	0	0x00000000	0b00000000000000000000000000000000
0	x12 (a2)	2147481120	0x7ffff620	0b01111111111111111111011000100000
0	x13 (a3)	0	0x00000000	0b00000000000000000000000000000000
0	x14 (a4)	0	0x00000000	0b00000000000000000000000000000000
0	x15 (a5)	0	0x00000000	0b00000000000000000000000000000000
0	x16 (a6)	0	0x00000000	0b00000000000000000000000000000000
0	x17 (a7)	0	0x00000000	0b00000000000000000000000000000000
0	x18 (s2)	2147481120	0x7ffff620	0b01111111111111111111011000100000

Input your RISC-V code here:

```
54  # This procedure clears 128 bits (16 bytes) of memory
55  # previously allocated to a dll_node
56  ########################################
57  FREE_DLLIST:
58
59  # Must clear the memory pointed to by x12 (16 bytes)
60  sw x0, 0(x12)           # Clears bytes 0-3
61  sw x0, 4(x12)           # Clears bytes 4-7
62  sw x0, 8(x12)           # Clears bytes 8-11
63  sw x0, 12(x12)          # Clears bytes 12-15
64
65  jalr x0, x1, 0          # jump and link back to main
66
67  END:
68  nop
```

Reset | Step | Run | CPU: 32 Hz

```
[line 47]: jalr x0, x1, 0
[line 18]: add x1, x0, x0
[line 19]: add x12, x0, x0
[line 22]: add x12, x18, x0
[line 23]: jal x1, FREE_DLLIST
```

Init Value	Register	Decimal	Hex	Binary
0	x0 (zero)	0	0x00000000	0b00000000000000000000000000000000
0	x1 (ra)	36	0x00000024	0b00000000000000000000000000100100
0	x2 (sp)	0	0x00000000	0b00000000000000000000000000000000
0	x3 (gp)	0	0x00000000	0b00000000000000000000000000000000
0	x4 (tp)	0	0x00000000	0b00000000000000000000000000000000
0	x5 (t0)	0	0x00000000	0b00000000000000000000000000000000
0	x6 (t1)	0	0x00000000	0b00000000000000000000000000000000
0	x7 (t2)	0	0x00000000	0b00000000000000000000000000000000
0	x8 (s0/fp)	0	0x00000000	0b00000000000000000000000000000000
0	x9 (s1)	0	0x00000000	0b00000000000000000000000000000000
0	x10 (a0)	0	0x00000000	0b00000000000000000000000000000000
0	x11 (a1)	0	0x00000000	0b00000000000000000000000000000000
0	x12 (a2)	2147481120	0x7ffff620	0b01111111111111111111011000100000
0	x13 (a3)	0	0x00000000	0b00000000000000000000000000000000
0	x14 (a4)	0	0x00000000	0b00000000000000000000000000000000
0	x15 (a5)	0	0x00000000	0b00000000000000000000000000000000
0	x16 (a6)	0	0x00000000	0b00000000000000000000000000000000
0	x17 (a7)	0	0x00000000	0b00000000000000000000000000000000
0	x18 (s2)	2147481120	0x7ffff620	0b01111111111111111111011000100000
0	x19 (s3)	0	0x00000000	0b00000000000000000000000000000000

Report Deliverable: Describe how you call the function calloc for the dllist struct, allocate the memory for the dllist struct, free the sllist struct main call, and detail a FREE_DLLIST procedure. You should show how your code allocates the 128 bits using calloc, as well as freeing the struct.

Set x12 to the address of the head pointer(x18) and then jump and link to CALLOC_DLLIST. In CALLOC_DLLIST, we stored x0 to 16 sequential bytes of memory starting at wherever x12 was pointing. Freeing the dllist similarly stores x0 to the 16 sequential bytes of memory starting at x12. After freeing the dllist struct main call, we added x0 and x0 and stored it in x18 to clear/zero the pointer. Then we branched to END using beq x0, x0, END.

Part 4.2 (dllist_step2.S)

Report Deliverable: Describe how you set up the for loop, and how you would insert the first node into the doubly linked list. You should show the steps of how the node is created, and how both head_ptr and tail_ptr are pointing to the only node.

Hint: Remember that, in a Doubly Linked List, that if there is only node, then both the head_ptr and tail_ptr point to that node. Also, remember the previous and next pointers must initially be NULL (whereas in the Singly Linked List, the next pointer must initially be NULL)

Memory Address		Decimal		Hex	Binary
0x7ffff620	head_ptr	2147481136		0x7ffff630	0b01111111111111111111011000110000
0x7ffff624	tail_ptr	2147481136		0x7ffff630	0b01111111111111111111011000110000
0x7ffff628		0		0x00000000	0b00000000000000000000000000000000
0x7ffff62c		0		0x00000000	0b00000000000000000000000000000000
0x7ffff630	node -> value	3		0x00000003	0b00000000000000000000000000000011
0x7ffff634	node -> prev_node	0	NULL	0x00000000	0b00000000000000000000000000000000
0x7ffff638	node -> next_node	0	NULL	0x00000000	0b00000000000000000000000000000000
0x7ffff63c		0		0x00000000	0b00000000000000000000000000000000

The for loop starts by checking the conditional to see if the block should be executed or skipped. It uses bge to compare x0 and x19. Meaning that if the value in x19 is less than or equal to zero, it will stop running the for loop by branching to “END_FOR”.

To insert the first node, the program determines that the head_ptr (x18→x7) is NULL. This means that the first node has not been inputted and, as such, the code does not take the “ELSE” path. Inside the IF statement, x18 (head_ptr) is stored in a temporary register and x28 is incremented by 16 bytes. This places the address in x28 at the location where the node→value will be stored for the first node we are about to insert. The following 4 addresses (or 16 bytes) are then cleared so as to account for the value, prev_node, next_node, and the segmentation between this current node and the eventual next node. Finally x7 and x28 are cleared, and y is decremented by one. The PC then jumps back to BEGIN_FOR. Eventually END_FOR is followed and x6 is cleared as well.

Note that prev_node and next_node are intentionally NULL and zero in the photo as there is no node that comes before it and no node that follows it yet.

Part 4.3 (dllist_step3.S)

Report Deliverable: Describe how you will insert the other nodes, including setting up the pointers to the previous and next nodes. Show examples of what the data memory looks like when as you step through the nodes.

Inserting the other nodes begins after the initial IF statement, which is reserved for inserting the first node. We start by loading the address of the head node from head_ptr into x28. We then get the next node by loading the location of x28 offset by 8 (2 addresses later of the head node). At this point, we check if the next_node is null and branch to END_WHILE if it is. We now set x29 = curr_ptr→next_node by loading an offset of 8(x28), effectively an offset of 8 for x29 since it was just loaded into x28. We then update the tail_ptr to x28, storing it in the obvious spot of x18(the head_ptr) offset by 4.

y=2

Memory Address		Decimal		Hex	Binary
0x7ffff620	head_ptr	2147481136		0x7ffff630	0b01111111111111111111011000110000
0x7ffff624	tail_ptr	2147481152		0x7ffff640	0b01111111111111111111011001000000
0x7ffff628		0		0x00000000	0b00000000000000000000000000000000
0x7ffff62c		0		0x00000000	0b00000000000000000000000000000000
0x7ffff630	node -> value	3		0x00000003	0b00000000000000000000000000000011
0x7ffff634	node -> prev_node	0	NULL	0x00000000	0b00000000000000000000000000000000
0x7ffff638	node -> next_node	2147481152		0x7ffff640	0b01111111111111111111011001000000
0x7ffff63c		0		0x00000000	0b00000000000000000000000000000000
0x7ffff640	node -> value	2		0x00000002	0b00000000000000000000000000000010
0x7ffff644	node -> prev_node	2147481136		0x7ffff630	0b01111111111111111111011000110000
0x7ffff648	node -> next_node	0	NULL	0x00000000	0b00000000000000000000000000000000
0x7ffff64c		0		0x00000000	0b00000000000000000000000000000000
0x7ffff650		0		0x00000000	0b00000000000000000000000000000000
0x7ffff654		0		0x00000000	0b00000000000000000000000000000000

y=1

Memory Address		Decimal	Hex	Binary
0x7ffff620	head_ptr	2147481136	0x7ffff630	0b01111111111111111111011000110000
0x7ffff624	tail_ptr	2147481168	0x7ffff650	0b01111111111111111111011001010000
0x7ffff628		0	0x00000000	0b00000000000000000000000000000000
0x7ffff62c		0	0x00000000	0b00000000000000000000000000000000
0x7ffff630	node -> value	3	0x00000003	0b00000000000000000000000000000011
0x7ffff634	node -> prev_node	0	0x00000000	0b00000000000000000000000000000000
0x7ffff638	node -> next_node	2147481152	0x7ffff640	0b01111111111111111111011001000000
0x7ffff63c		0	0x00000000	0b00000000000000000000000000000000
0x7ffff640	node -> value	2	0x00000002	0b00000000000000000000000000000010
0x7ffff644	node -> prev_node	2147481136	0x7ffff630	0b01111111111111111111011000110000
0x7ffff648	node -> next_node	2147481168	0x7ffff650	0b01111111111111111111011001010000
0x7ffff64c		0	0x00000000	0b00000000000000000000000000000000
0x7ffff650	node -> value	1	0x00000001	0b00000000000000000000000000000001
0x7ffff654	node -> prev_node	2147481152	0x7ffff640	0b01111111111111111111011001000000
0x7ffff658	node -> next_node	0	0x00000000	0b00000000000000000000000000000000
0x7ffff65c		0	0x00000000	0b00000000000000000000000000000000
0x7ffff660		0	0x00000000	0b00000000000000000000000000000000
0x7ffff664		0	0x00000000	0b00000000000000000000000000000000
0x7ffff668		0	0x00000000	0b00000000000000000000000000000000

NULL

NULL

y=0

Memory Address		Decimal	Hex	Binary
0x7ffff620	head_ptr	2147481136	0x7ffff630	0b01111111111111111111011000110000
0x7ffff624	tail_ptr	2147481184	0x7ffff660	0b01111111111111111111011001100000
0x7ffff628		0	0x00000000	0b00000000000000000000000000000000
0x7ffff62c		0	0x00000000	0b00000000000000000000000000000000
0x7ffff630	node -> value	3	0x00000003	0b00000000000000000000000000000011
0x7ffff634	node -> prev_node	0	0x00000000	0b00000000000000000000000000000000
0x7ffff638	node -> next_node	2147481152	0x7ffff640	0b01111111111111111111011001000000
0x7ffff63c		0	0x00000000	0b00000000000000000000000000000000
0x7ffff640	node -> value	2	0x00000002	0b00000000000000000000000000000010
0x7ffff644	node -> prev_node	2147481136	0x7ffff630	0b01111111111111111111011000110000
0x7ffff648	node -> next_node	2147481168	0x7ffff650	0b01111111111111111111011001010000
0x7ffff64c		0	0x00000000	0b00000000000000000000000000000000
0x7ffff650	node -> value	1	0x00000001	0b00000000000000000000000000000001
0x7ffff654	node -> prev_node	2147481152	0x7ffff640	0b01111111111111111111011001000000
0x7ffff658	node -> next_node	2147481184	0x7ffff660	0b01111111111111111111011001100000
0x7ffff65c		0	0x00000000	0b00000000000000000000000000000000
0x7ffff660		0	0x00000000	0b00000000000000000000000000000000
0x7ffff664	node -> value	2147481168	0x7ffff650	0b01111111111111111111011001010000
0x7ffff668	node -> prev_node	0	0x00000000	0b00000000000000000000000000000000
0x7ffff66c	node -> next_node	0	0x00000000	0b00000000000000000000000000000000

NULL

Part 4.4 (dllist_step4.S)

Report Deliverable: Describe how you will find and free every dll_node*, including seeking and freeing the nodes.

We load the head node and the next subsequent node into curr_ptr and next_ptr, respectively accomplishing the following C code:

```
sll_node* curr_ptr = the_list→head_ptr;
sll_node* next_ptr = curr_ptr→next_ptr;
```

We then create a “FREE_NODE_LOOP” which will iterate through all the nodes in memory, freeing them as necessary with sw instructions. We continue to loop until the curr_ptr(x7) is NULL (equal to x0). We then free the curr_ptr (its subsequent 16 bytes). We then update the pointer to next_ptr and set the next_ptr equal to the curr_ptr→next_ptr through x7 offset by 8 accomplishing the following C code:

```
curr_ptr = next_ptr;
next_ptr = curr_ptr→next_ptr;
```

We then continue the loop until all nodes have been cleared from memory.

Part 4.5 (dllist_final.S)

Report Deliverable: Show how you would find and delete one element in the Linked List based on the value. Note how this is called before the free() in main. This means that the find code should delete one node, and then free the other nodes after the call.

We will load word a current pointer(x7) with the head of the list and then run the equivalent of a while loop until we find the pointer that contains the value or the pointer points to null(using beq x7, x0, END_DELETE_NODE_LOOP). If the pointer that contains the value is found then there are a series of if statements depending on if the pointer is the head, somewhere in the middle, the tail, or if the head pointer is equal to the tail pointer.

If the head pointer is equal to the tail pointer, then we set both of them to zero using store word and x0. Else if the pointer is the head, then we use lw and sw to make the head pointer point to the curr→next, and then we make the head pointer's prev pointer's value zero. Else if the pointer is the tail, then the tail pointer becomes the curr_ptr→prev and then the tail pointer→next value is set to zero using lw's and sw's like in the previous else if. Finally, the else statement is for if the pointer is a middle node. Thus, we use a series of lw's, addi's, and sw's to 'bridge the gap' that the deletion will make. The curr→prev→next is set to curr→next and the curr→next→prev is set to curr→prev. At the end of each of the if and elif statements, there is a beq x0, x0, END_DELETE_IF. This clears all of the registers that were used and also deletes the node from memory.

If the value is not found on the current pointer, then curr = curr→next. We did this by getting the curr→next from memory (lw x28, 8(x7)) and then setting x7 equal to it with add x7, x28, x0.

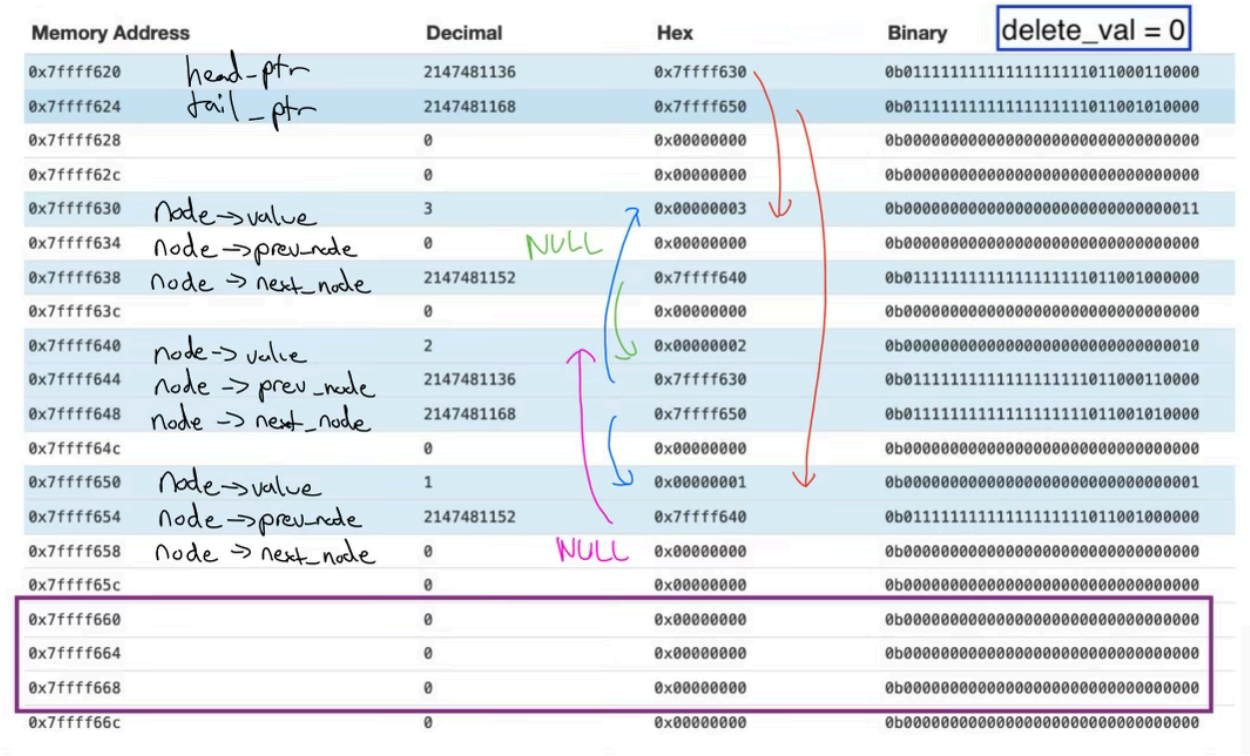
See subsequent pages for images.

delete_val = 3

Memory Address		Decimal	Hex	Binary
0x7ffff620	head_ptr	2147481152	0x7ffff640	0b01111111111111111111011001000000
0x7ffff624	tail_ptr	2147481184	0x7ffff660	0b01111111111111111111011001100000
0x7ffff628		0	0x00000000	0b00000000000000000000000000000000
0x7ffff62c		0	0x00000000	0b00000000000000000000000000000000
0x7ffff630		0	0x00000000	0b00000000000000000000000000000000
0x7ffff634		0	0x00000000	0b00000000000000000000000000000000
0x7ffff638		0	0x00000000	0b00000000000000000000000000000000
0x7ffff63c		0	0x00000000	0b00000000000000000000000000000000
0x7ffff640	node->value	2	0x00000002	0b00000000000000000000000000000010
0x7ffff644	node->prev_node	0 NULL	0x00000000	0b00000000000000000000000000000000
0x7ffff648	node->next_node	2147481168	0x7ffff650	0b01111111111111111111011001010000
0x7ffff64c		0	0x00000000	0b00000000000000000000000000000000
0x7ffff650	node->value	1	0x00000001	0b00000000000000000000000000000001
0x7ffff654	node->prev_node	2147481152	0x7ffff640	0b01111111111111111111011001000000
0x7ffff658	node->next_node	2147481184	0x7ffff660	0b01111111111111111111011001100000
0x7ffff65c		0	0x00000000	0b00000000000000000000000000000000
0x7ffff660	node->value	0	0x00000000	0b00000000000000000000000000000000
0x7ffff664	node->prev_node	2147481168	0x7ffff650	0b01111111111111111111011001010000
0x7ffff668	node->next_node	0 NULL	0x00000000	0b00000000000000000000000000000000
0x7ffff66c		0	0x00000000	0b00000000000000000000000000000000

delete_val =2

Memory Address		Decimal	Hex	Binary
0x7ffff620	head_ptr	2147481136	0x7ffff630	0b01111111111111111111011000110000
0x7ffff624	tail_ptr	2147481184	0x7ffff660	0b01111111111111111111011001100000
0x7ffff628		0	0x00000000	0b00000000000000000000000000000000
0x7ffff62c		0	0x00000000	0b00000000000000000000000000000000
0x7ffff630	node->value	3	0x00000003	0b00000000000000000000000000000011
0x7ffff634	node->prev_node	0 NULL	0x00000000	0b00000000000000000000000000000000
0x7ffff638	node->next_node	2147481168	0x7ffff650	0b01111111111111111111011001010000
0x7ffff63c		0	0x00000000	0b00000000000000000000000000000000
0x7ffff640		0	0x00000000	0b00000000000000000000000000000000
0x7ffff644		0	0x00000000	0b00000000000000000000000000000000
0x7ffff648		0	0x00000000	0b00000000000000000000000000000000
0x7ffff64c		0	0x00000000	0b00000000000000000000000000000000
0x7ffff650	node->value	1	0x00000001	0b00000000000000000000000000000001
0x7ffff654	node->prev_node	2147481136	0x7ffff630	0b01111111111111111111011000110000
0x7ffff658	node->next_node	2147481184	0x7ffff660	0b01111111111111111111011001100000
0x7ffff65c		0	0x00000000	0b00000000000000000000000000000000
0x7ffff660	node->value	0	0x00000000	0b00000000000000000000000000000000
0x7ffff664	node->prev_node	2147481168	0x7ffff650	0b01111111111111111111011001010000
0x7ffff668	node->next_node	0 NULL	0x00000000	0b00000000000000000000000000000000
0x7ffff66c		0	0x00000000	0b00000000000000000000000000000000


delete_val = 0
Memory Address
Decimal
Hex
Binary
0x7ffff620
head_ptr
2147481136
0x7ffff630
0b01111111111111111111011000110000
0x7ffff624
tail_ptr
2147481168
0x7ffff650
0b01111111111111111111011001010000
0x7ffff628
0
0x00000000
0b00000000000000000000000000000000
0x7ffff62c
0
0x00000000
0b00000000000000000000000000000000
0x7ffff630
node->value
3
0x00000003
0b00000000000000000000000000000011
0x7ffff634
node->prev_node
0
NULL
0x00000000
0b00000000000000000000000000000000
0x7ffff638
node -> next_node
2147481152
0x7ffff640
0b01111111111111111111011001000000
0x7ffff63c
0
0x00000000
0b00000000000000000000000000000000
0x7ffff640
node -> value
2
0x00000002
0b00000000000000000000000000000010
0x7ffff644
node -> prev_node
2147481136
0x7ffff630
0b01111111111111111111011000110000
0x7ffff648
node -> next_node
2147481168
0x7ffff650
0b01111111111111111111011001010000
0x7ffff64c
0
0x00000000
0b00000000000000000000000000000000
0x7ffff650
node->value
1
0x00000001
0b00000000000000000000000000000001
0x7ffff654
node->prev_node
2147481152
0x7ffff640
0b01111111111111111111011001000000
0x7ffff658
node -> next_node
0
NULL
0x00000000
0b00000000000000000000000000000000
0x7ffff65c
0
0x00000000
0b00000000000000000000000000000000
0x7ffff660
0
0x00000000
0b00000000000000000000000000000000
0x7ffff664
0
0x00000000
0b00000000000000000000000000000000
0x7ffff668
0
0x00000000
0b00000000000000000000000000000000
0x7ffff66c
0
0x00000000
0b00000000000000000000000000000000