1. ***Hardware Design ISA-bout Tradeoffs (15 points):*** Consider the following three properties:

   - *Instruction Count*
   - *Cycle Time*
   - *Hardware Complexity* (i.e .the number of logic gates required to implement the hardware or the cost to verify the design - the hardware is more difficult to design/verify)

For the following design decisions, indicate how these three properties could be impacted. Your answer for each should be that it either ***could increase, could decrease, or will stay the same.*** You may only circle one answer per property. The first one is done for you as an example.

Example: ***Design Decision***: Use a carry lookahead adder instead of a ripple carry adder.

• *Instruction Count*	could increase	could decrease	(will stay the same)
• *Cycle Time*	could increase	(could decrease)	will stay the same
• *Hardware Complexity*	(could increase)	could decrease	will stay the same

a. ***Design Decision***: Hardware that is designed to use a variable length instruction format instead of a fixed length instruction format.

• *Instruction Count*	could increase	(could decrease)	(will stay the same)
• *Cycle Time*	(could increase)	(could decrease)	will stay the same
• *Hardware Complexity*	(could increase)	could decrease	will stay the same

b. ***Design Decision***: Hardware that is designed with a smaller number of registers instead of a larger number of registers.

• *Instruction Count*	(could increase)	could decrease	will stay the same
• *Cycle Time*	could increase	(could decrease)	will stay the same
• *Hardware Complexity*	could increase	(could decrease)	will stay the same

c. ***Design Decision***: Hardware based on a load-store machine instead of a register-memory machine.

• *Instruction Count*	(could increase)	could decrease	will stay the same
• *Cycle Time*	could increase	(could decrease)	(will stay the same)
• *Hardware Complexity*	could increase	(could decrease)	will stay the same

+9

2. ***Performance Anxiety (25 points)***: Suppose on a particular load-store machine-based processor there are four types of operations - loads, stores, branches, and ALU operations (i.e. adds, subtracts, multiplies). This will be implemented by some other datapath – **not** the single-cycle datapath we covered in class. Loads take 5 cycles, stores take 4 cycles, branches take 3 cycles, and ALU operations take 4 cycles. The architecture is not pipelined – so no instruction latency is overlapped – instructions are executed one at a time, in order. Answer the questions below – show your work.

a. What is the CPI for an application with one million instructions that is 30% loads, 10% stores, 40% ALU operations, and 20% branches? (*5 points*).

CPI: 4.1

CPI = Clock cycles / Instruction count

$\frac{(0.3)(5)(1\times10^6) + (0.5)(4)(1\times10^6) + (0.2)(3)(1\times10^6)}{1\times10^6} = (0.3)(5) + (0.5)4 + (0.2)3$

$1.5 + 2 + 0.6$

b. Now suppose that 50% of loads and 50% of stores can be eliminated by a new algorithm that makes more effective use of memory. But the cost is increased computation – 25% more ALU operations are required. What is the new CPI?

CPI: 3.94

total num of ins: $1\times10^6 + (.25)(.4)1\times10^6 - (.5)(.3)1\times10^6 - (.5)(.1)1\times10^6$

$= 1\times10^6(1 + 0.1 - 0.15 - 0.05)$

$= 900{,}000$

$5(0.5)(0.3)1\times10^6 + 4(0.5)(0.1)1\times10^6 + 4(1.25)(0.4)(1\times10^6) + 3(0.2)(1\times10^6) = \frac{0.75+0.2+2+0.6}{0.9} = 3.94$

c. As an alternative (i.e. we keep the original algorithm from part a) suppose we try instead to increase the speed of the processor clock. The clock frequency can be made 50% faster but the latency of each operation will also require 50% more cycles. Is the optimization worth it? Calculate the speedup or slowdown over the original execution time of the algorithm from part a on the original processor clock speed.

Speedup or Slowdown: yes speedup 1.33
(relative to part a)

new CPU Time = ins count * CPI / clock rate. $= CPI_{new} = (1.5)(4.1) = 6.15 \times$ (clock cycle time)

$(6.15)(0.5 t_{cc}) = 3.075\ t_{cc}$

not equivalent to

old CPU Time = ins count * CPI / clock rate $= (CPI_{old} = 4.1) \times$ (clock cycle time)

$(4.1)(t_{cc})$

-2

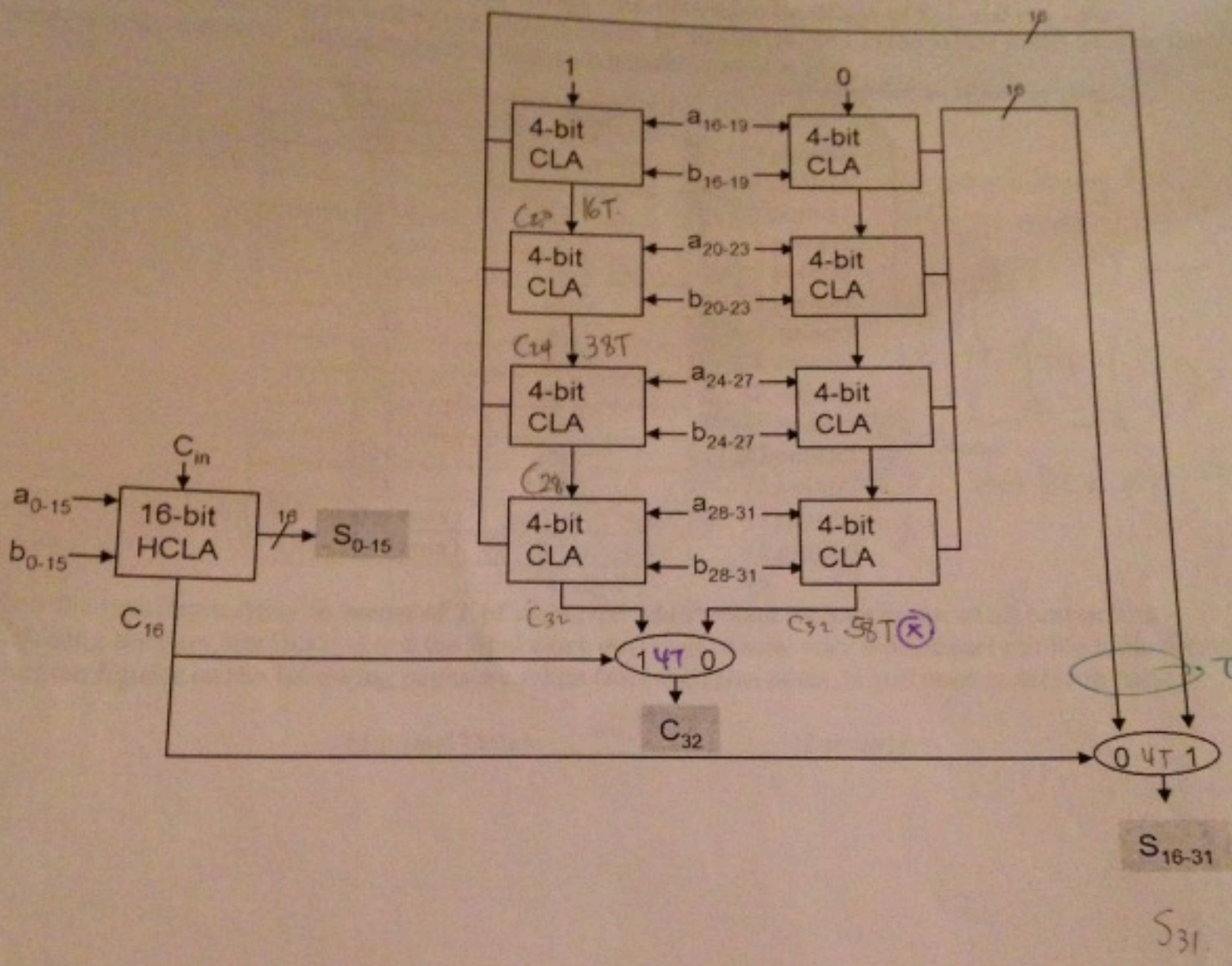
+23

3. ***Give Up the Funk (30 points)***: Consider the single-cycle processor implementation. Your task will be to augment this datapath with a new instruction: the *funkyb* instruction. This instruction will be an R-type instruction, and will have the following effect:

```
if (R[rs]<R[rt])
        PC=PC+4+R[rs]          // Note that these two statements
        R[rd]=R[rt]            // will be concurrent.
else
        PC=PC+4+R[rt]          // Note that these two statements
        R[rd]=R[rs]            // will be concurrent.
```

Implement *funkyb* on the **single cycle datapath**. Use the R-type instruction format – so this instruction will have the same opcode as all other R-types. Use a unique function field to modify the ALU controller to implement this instruction, not the main controller.

Implement your solution on the following two pages. All other instructions must still work correctly after your modifications. You should not add any new ALUs, register file ports, or ports to memory.


1
0
4-bit CLA
4-bit CLA
a_{16-19}
b_{16-19}
16T
4-bit CLA
4-bit CLA
a_{20-23}
b_{20-23}
38T
4-bit CLA
4-bit CLA
a_{24-27}
b_{24-27}
4-bit CLA
4-bit CLA
a_{28-31}
b_{28-31}
C_{in}
a_{0-15}
b_{0-15}
16-bit HCLA
16
S_{0-15}
C_{16}
1 4T 0
C_{32}
0 4T 1
S_{16-31}
S_{31}

4. ***Putting the CLA into UCLA (30 points)***: Assume for the rest of this problem that all logic gates have the following delays:

Fan In	Delay
1	T
2	2T
3	3T
4	5T
5	7T
6	10T
7 or more	2T x fan-in

So a 2-input AND gate would have delay T and a 4-input OR gate would have delay 3T.

For simplicity, assume that mux's have delay 4T regardless of fan-in.

We will create a 32-bit adder out of some building blocks we've covered in class. We will use the 4-bit CLA that we covered in class as one basic building block of this design. And we will use it (as we did in class) to make 16-bit hierarchical CLAs (HCLA) which will be our other building block. But instead of connecting these in series to make a 32-bit adder, we will use carry select to speed up the 32-bit adder. The design will look as follows (be sure to note where we are using CLAs and where we are using HCLAs):

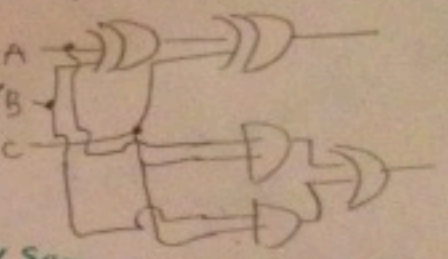
Your task is to find the maximal delay of this design – i.e. determine the delays of $S_{0:31}$ and C_{32} – the maximal delay of these outputs will be the maximal delay of the design. Fill in the values in the table on the following page to receive full credit (and to help with possible partial credit).

Output	Delay
G0	2T
P0	2T
Gα	~~10T~~
Pα	7T
C12	~~18T~~ (13T+5T)
C15	~~32T~~
C16	~~22T~~
S15	~~36T~~
C20	16T
S19	~~46T~~
C24	~~38T~~
C31	54T
C32 (after mux)	58T+4T=62T
S31 (after mux)	54T+4T=62T

Work is labeled on following page.

i (2 points)
(2 points)
(2 points)
(2 points)
OK (2 points)
(2 points)
(2 points)
(2 points)
(2 points)
(2 points) OK
(2 points)
viii (2 points)
⊗ (2 points)
(2 points)

using following 1bit full adder:



same mistake as S15

Find the maximum delay **in terms of T** of the 32-bit adder – take the maximum of all output bits – including the sum bits (S_0-S_{31}) and the final carry out (C_{32}). Show your work clearly in the table above. The two figures on the following pages are taken from the class notes, if you need to refer to them.

Maximal Delay: 62T _____ *(2 points)*