Del A:

1. Den används för att peka vart i minnet som nästa instruktion finns (2p). Den ökas när nästa instruktion ska exekveras, den ändras vid hopp (1p/varje)
2. Stacken fungerar som en hög med tallrikar, dvs. data som lagrades senast är det som först returneras, Last In First Out (LIFO) (2p). Stacken i 68000 växer mot låga adresser (1p). MOVE (SP)+, D2 (1p).
   MOVE D2, -(SP) (1p).
3. a. Von Neumann arkitekturen kännetecknas av att den har en systembuss där både data och instruktioner skickas (3p). Dettar med för att en instruktion kräver flera klockcykler för att exekveras, endast 25% av tiden används exekveringsenheten (1p). Exekvering av instruktion: a. Läs operationsord, b. Läs operand, c. Utför operation, d. Skriv resultat. (2p)
   b. Pipeline, Cacheminne, Superpipeline, Superskalär (1p/varje). Pipeline ger instruktionsparallellitet, dvs. flera exekveringsteg utförs samtidigt (1p). Cacheminnet minnskar accesstiden till minnet (1p). Med en superpipeline görs mindre under varje klockcykel, dvs. möjliggör högre frekvens (1p). Med superskalärarkitektur kan flera instruktioner exekveras samtidigt (1p).
4. Adressbuss, Databuss, och Kontrollbus (3p). Adressbussen används för att peka ut den minnesposition som ska läsas/skrivas (1). På databussen skickas data från/till processorn (1p). Kontrollbussen används för att synkronisera processorn med externa enheter (tex. Minne, periferienhet) (1).
5. Via register, via fast minnesposition, och via stacken (3p). Register: +Snabb, -begränsat antal register (1p). Fast minnesposition: +Obegränsat antal, -statiskt (stöder inte rekursion och dynamisk minneshantering) (1p). Stack: +Obegränsat antal parametrar, dynamisk, - inte lika snabb som register (1p).
6. (1p/varje)

		C	Z	D1
a. ADDI.B	#11,D1	0	0	$B3
b. SUBI.B	#$70,D1	0	0	$38
c. ADDI.B	#$A0,D1	1	0	$48
d. SUBI.B	#$64,D1	0	0	$44
e. ASL.B	#6,D1	0	1	0
f. LSR.B	#2,D1	0	0	$2A

7. RISC = Reduced Instruction Set Computer (1p). CISC = Complex Instruction Set Computer (1p). RISC har få instruktioner => liten HW implementering (snabb impl. och enkel att konstruera) (1p). CISC = Mycket många instruktioner, mycket HW, komplex att implementera. (1p)
8. a. När ett undantag inträffar så: (I) Processorn kopierar SR->SR_tmp. (II) Sätter S flaggan till 1. (III) Vektornumret bestäms. (IV) PC -> (SP), SR_tmp -> (SP). (V) Det nya PC-värdet laddas från undantagsvektorn, och undantagsrutinen startas.
   b. Mha RTE
   c. Division med noll, busserror, adresserror, interrupt, TRAP (1p/varje)

Del B:

9.
```
start:      MOVE.W   #20, D3
next:       ADD.W    D1,D0
            ADD.W    D2,D0
            MOVE.W   D1,D2
            MOVE.W   D0,D1
            SUBQ.W   #1,D3
            BNE      next
```

10.
```
OUT_1       EQU      $FF0000
IN_1        EQU      $FF0002
            ORG      $4000
start:      MOVE.B   IN_1,D0
            CLR.B    D1
            BCLR.B   #7,OUT_1
next_0:     SUBQ.B   #1,D1
            CMP.B    D0,D1
            BNE      next_0
            BSET.B   #7,OUT_1
next_1:     SUBQ.B   #1,D1
            NOP
            BNE      next_1
            BRA      start
```

11. $n_{comp} = 2^{20}/(2^{18}*0.5) = 8$ stycke minneskomponenter
    CS_{bank_0} = ~A19*~A18*AS
    CS_{bank_1} = ~A19* A18*AS
    CS_{bank_2} = A19*~A18*AS
    CS_{bank_3} = A19* A18*AS

12. A23-A20 via en ELLER-grid in på en enable-signal, AS in på enable signal.
    IO = A15*…*A8
    CS_ROM = O0*O1
    CS_RAM1 = O2*O3*O4
    CS_RAM2 = O5*O6*…*OB
    CS_IO0 = OC * IO
    CS_IO1 = OD * IO
    CS_IO2 = OE * IO
    CS_IO3 = OF * IO

13.
```
copy_data:  MOVEM.L  D0/A0-A1, -(SP)
            MOVEA.L  16(SP), A0
            MOVEA.L  20(SP), A1
            MOVE.W   24(SP), D0
next        MOVE.W   (A0)+,(A1)+
            SUBQ.W   #1,D0
            BNE      next
            MOVEM.L  (SP)+,D0/A0-A1
            RTS
```