

Lösningsförslag till Tenta i Mikrodator 040117

1. Vilka register finns det i processorn och vad används dessa till?

D0 till D7: Dataregister som används för beräkningar

A0 till A6: Adressregister som används till att lagra pekare till minnet

A7: pekar på stacken och är dubblerad till USP och SSP som används beroende på vilken mod processorn befinner sig i, *user mode* eller *supervisor mode*.

PC: Programräknaren pekar på den adress i minnet som nästa instruktion skall hämtas ifrån.

SR: statusregistret styr några av processorns funktioner, avbrottsmask, trace och användarmod. Status från aritmetiska operationer indikeras i ett antal statusbitar.

2. Nämn och beskriv kortfattat två sätt att adressera data i ett cacheminne.

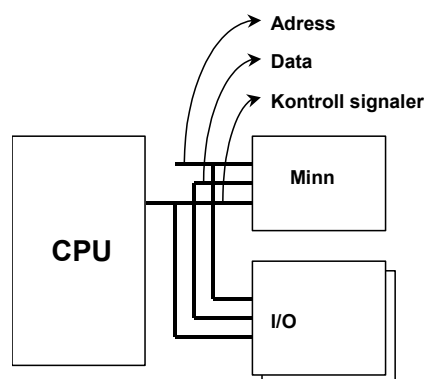
Direktmappad cache: Cacheminnet är indelat i N stycken lika stora linjer. Primärminnet är indelat i K stycken block. Ett sådant block består i sin tur av N stycken linjer. En linje n i primärminnet kan endast lagras i motsvarande linje n i cacheminnet. Därav direktmappat.

Associativ cache: En fullt associativ cache är precis som för den direktmappade indelat i N stycken linjer. En linje i primärminnet kan för en associativ adressering lagras i godtycklig linje i cacheminnet. Denna flexibla adressering gör också att sökningen i det associativa cacheminnet blir mer komplext och dyrare att implementera.

3. Vilken uppgift har UDS* och LDS* ?

UDS* och LDS* är övre respektive nedre datastrobe. LDS indikerar att den minst signifikanta byten skall adresseras. UDS indikerar att den mest signifikanta byten skall adresseras.

4. Vad står DMA för samt beskriv hur DMA fungerar?



DMA står för Direct Memory Access och är en metod för processorn att synkronisera mot I/O-enheter. När en I/O-enhet behöver överföra data till minnet så begär den att få ta över kontrollen över systembussen, dvs bli busmästare. Processorn lämnar då över kontrollen till I/O som nu kan direkt överföra data till minnet utan att gå via processorn. Efter avslutad dataöverföring lämnas kontrollen över systembussen tillbaka till processorn.

5. Antag att register D1 innehåller \$00000099. Ange D1, C-flaggans samt Z- flaggans värde efter att instruktionerna nedan har exekverats. Observera att a till f är enskilda instruktioner och inte ett program.

a) **EORL.B #\$66,D1**

$$\begin{array}{r} \$99 = 1001\ 1001 \\ \$66 = 0110\ 0110 \\ \text{XOR} \\ \hline \$FF\ 1111\ 1111 \end{array}$$

→ C=0 , Z = 0, D1=\$000000FF

b) **ADDL.B #\$66,D1**

$$\begin{array}{r} \$99 = 1001\ 1001 \\ \$66 = 0110\ 0110 \\ \text{ADD} \\ \hline \$FF\ 0\ 1111\ 1111 \end{array}$$

→ C=0 , Z = 0, D1=\$000000FF

c) **ADDL.B #\$67,D1**

$$\begin{array}{r} \$99 = 1001\ 1001 \\ \$67 = 0110\ 0111 \\ \text{ADD} \\ \hline \$100 = 1\ 0000\ 0000 \end{array}$$

→ C=1 , Z = 1, D1=\$00000000

d) **SUBL.B #\$66,D1**

$$\begin{array}{r} \$99 = 1001\ 1001 \\ \$66 = 0110\ 0110 \\ \text{SUB} \\ \hline \$033 = 0\ 0011\ 0011 \end{array}$$

→ C=0 , Z = 0, D1=\$00000033

e) **ASR.B #2,D1**

$$\begin{array}{r} \$99 = 1001\ 1001 \\ \text{ASR } 2 \\ \hline \$E6 = 1110\ 0110 \end{array}$$

→ C=0 , Z = 0, D1=\$000000E6

f) **ROL.B #4,D1**

$$\begin{array}{r} \$99 = 1001\ 1001 \\ \text{ROL } 4 \\ \hline \$99 = 1001\ 1001 \end{array}$$

→ C=1 , Z = 0, D1=\$00000099

6. Funktionsanrop i assembler. En funktion i C är deklarerade som följande:

```
void printChar(int a, int b);
```

Parametrarna a och b ska skickas till funktionen via stacken.

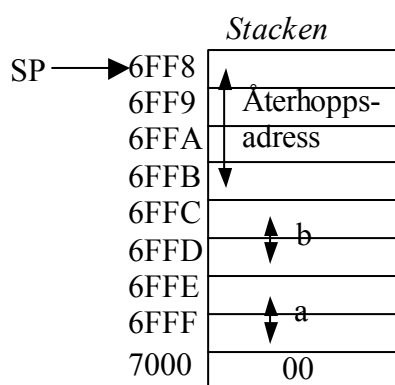
- a) Rita stackens innehåll efter hoppet till subrutinen printChar, precis innan första instruktionen i subrutinen exekverats. Stackpekaren pekar på adress \$7000 innan anropet. Anropet till funktionen i assembler görs enligt följande:

Funktionsanrop i C

```
PrintChar(a,b);
```

Funktionsanrop i assembler

```
MOVE.W a, -(A7)
MOVE.W b, -(A7)
BSR printChar
ADDA.L #4, A7
```



b) Vilken funktion har instruktionen ADDA.L #4,A7 ?

Instruktionen "tar bort" parametrarna a och b från stacken. (Med "tar bort" i detta sammanhang menas att stackpekaren justeras +4.)

7. Adresseringsmetoder: Vad kommer D2 och A3 att innehålla efter att instruktionen har exekverats. Innehållet i minnet är specificerat av tabellen nedan.

Instruktionerna är oberoende av varandra (inget program). D2 har värdet \$00000000 och A3 \$00004004 innan varje instruktion exekveras.

- MOVE.L \$4004, D2
- MOVE.W 4(A3), D2
- MOVE.L -(A3), D2
- MOVE.B -(A3), D2
- MOVE.W -2(A3), D2
- MOVE.W #\$4004, D2
- MOVE.B (A3)+, D2
- MOVE.L (A3)+, D2

Adress (Hexadecimal)	Innehåll (Hexadecimal)
4000	77
4001	21
4002	C3
4003	FD
4004	00
4005	14
4006	99
4007	74
4008	A6
4009	AA

Deluppg.	Instruktion	D2	A3
A	MOVE.L \$4004, D2	\$00149974	\$00004004
B	MOVE.W 4(A3), D2	\$0000A6AA	\$00004004
C	MOVE.L -(A3), D2	\$7721C3FD	\$00004000
D	MOVE.B -(A3), D2	\$000000FD	\$00004003
E	MOVE.W -2(A3), D2	\$0000C3FD	\$00004004
F	MOVE.W #\$4004, D2	\$00004004	\$00004004
G	MOVE.B (A3)+,D2	\$00000000	\$00004005
H	MOVE.L (A3)+,D2	\$00149974	\$00004008

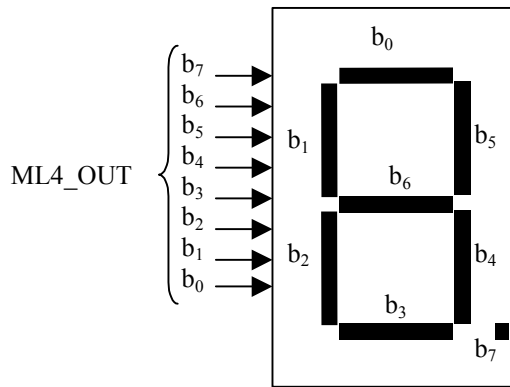
8. **Assemblerprogrammering:** Skriv en subrutin i 68000-assembler som skriver ut ett tal (0-15) till en display. På displayen presenteras talet som en hexadecimal siffra (0-F). Displayen är av samma typ som på labbkorten, dvs. en 7-segment display.

Displayen är ansluten till ML4_OUT (porten är definierad). Varje bit i ML4_OUT är ansluten till en lysdiod i 7-segment displayen. Kopplingen mellan bitarna i ML4_OUT och lysdioderna är definierad i figuren nedan (T ex. b_0 är kopplad till den översta vågräta dioden). b_7 är kopplad till punkten på 7-segmentdisplayen, så den sätter ni alltid till noll.

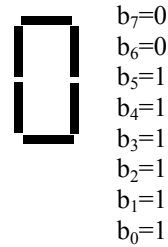
In: ML4_OUT = Talet som ska skrivas ut till 7-segmentdisplayen (tal mellan 0-16)

Ut: Inget ska returneras från subrutinen.

Exempel: Nedre byten på D0 = %00110000 ska ge en etta på displayen och ML4_OUT = %00111111 ger en nolla.



Ex. för en nolla
ML4_OUT = %00111111



* Subroutine: writeToSevenSegment

*

* Input parameters: D0.B number to be written

* Output: none

writeToSevenSegment:

MOVEM.L D0/A0,-(SP)
BRA startOfSub

*A0, D0 will be used and copied to the stack

segmentLUT:

SEGMENTS76543210*****

OUT_0 DC.B %00111111
OUT_1 DC.B %00110000
OUT_2 DC.B %01101101
OUT_3 DC.B %11111001
OUT_4 DC.B %01110010
OUT_5 DC.B %01011011
OUT_6 DC.B %01011111
OUT_7 DC.B %00110001
OUT_8 DC.B %01111111
OUT_9 DC.B %01111011
OUT_A DC.B %01110111
OUT_B DC.B %01011110
OUT_C DC.B %00001111
OUT_D DC.B %01111100
OUT_E DC.B %01001111
OUT_F DC.B %01000111

ALIGN

startOfSub:

ANDI.L #\$F,D0
LEA segmentLUT,A0
ADD.A L D0,A0
MOVE.B (A0),ML4_OUT

MOVEM.L (SP)+,D0/A0
RTS

* Only four bit parameter

* A0 points at start of table

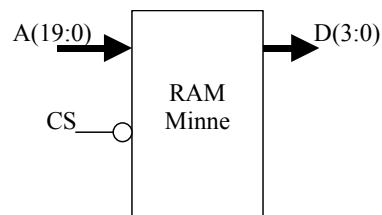
* Add parameter to start of table

* Copy the 7-seg code that A0 points at

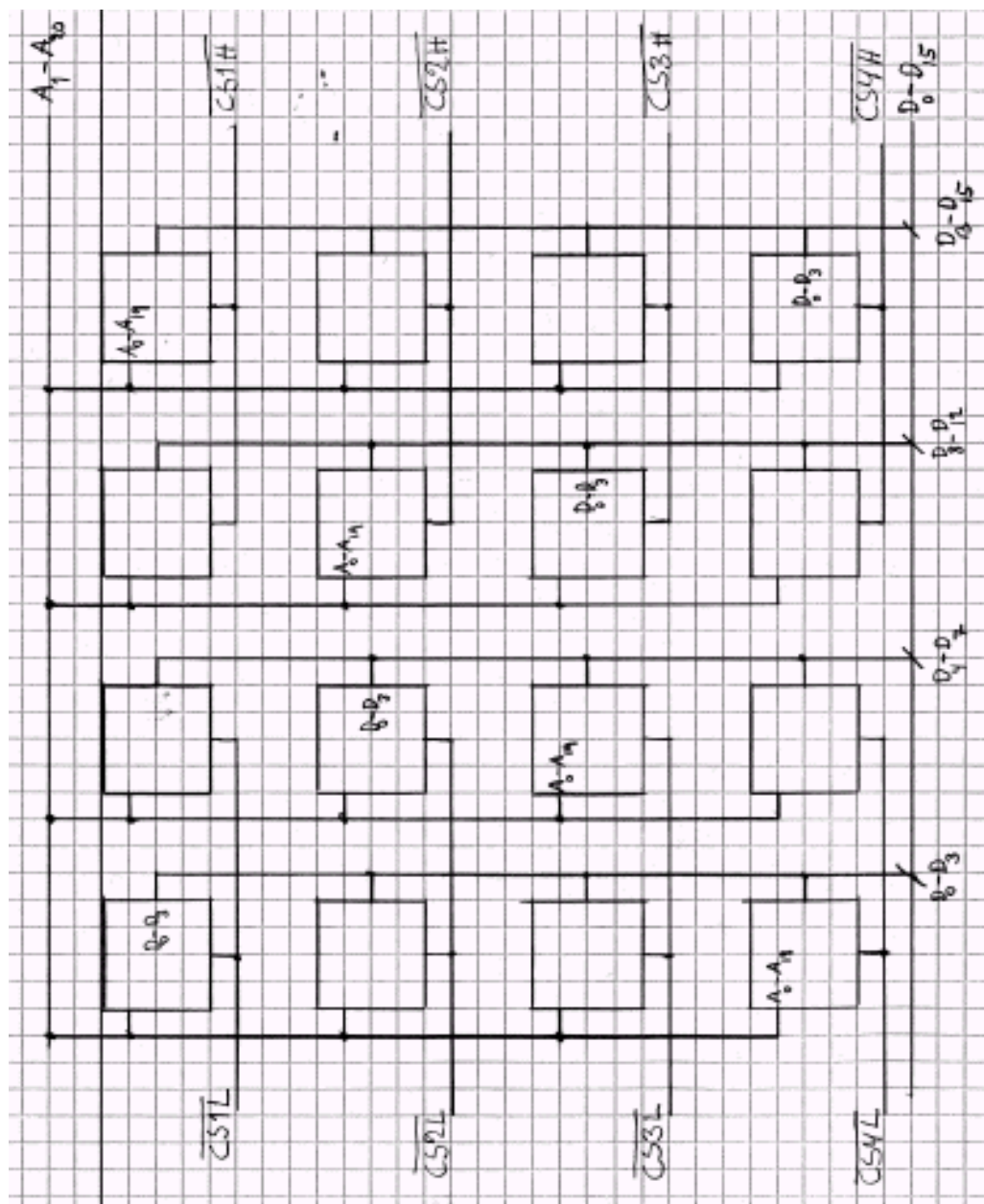
* Retrieve old value of A0,D0

9. Sätt samman en minnesbank som är konstruerad av flera mindre minnesobjekt (se komponenten i figuren). Signaler till minnesbanken ska vara A(adress), D(data) samt CS*. Minneskapslarna som används för att konstruera minnesbanken har samma signaler som den färdiga minnesbanken har. Där CS* signalen är aktivt låg. Du ska indikera vilka adress/data-signaler som ska kopplas till de olika minneskapslarna.

Minnesbanken ska vara 8Mbytes stort organiserat med en 16 bitars databuss. Minnesbanken sätts samman med hjälp av minneskomponenter som har 4-bitars databuss och 20 bitars adressbuss.



$$\begin{aligned}
 m_{\text{comp}} &= \frac{S}{2^{n_{\text{adress comp}}} \cdot \frac{n_{\text{data comp}}}{8}} = \frac{2^{23}}{2^{20} \cdot \frac{4}{8}} = \frac{2^3}{1/2} \\
 &= 2^4 = \underline{\underline{16 \text{ kapslar}}} \\
 m_{\text{kp6}} &= \frac{n_{\text{data bus}}}{n_{\text{data comp}}} = \frac{16}{4} = \underline{\underline{4 \text{ kapslar/bank}}} \\
 m_{\text{bank}} &= m_{\text{comp}} / m_{\text{kp6}} = \frac{16}{4} = \underline{\underline{4 \text{ banker}}}
 \end{aligned}$$

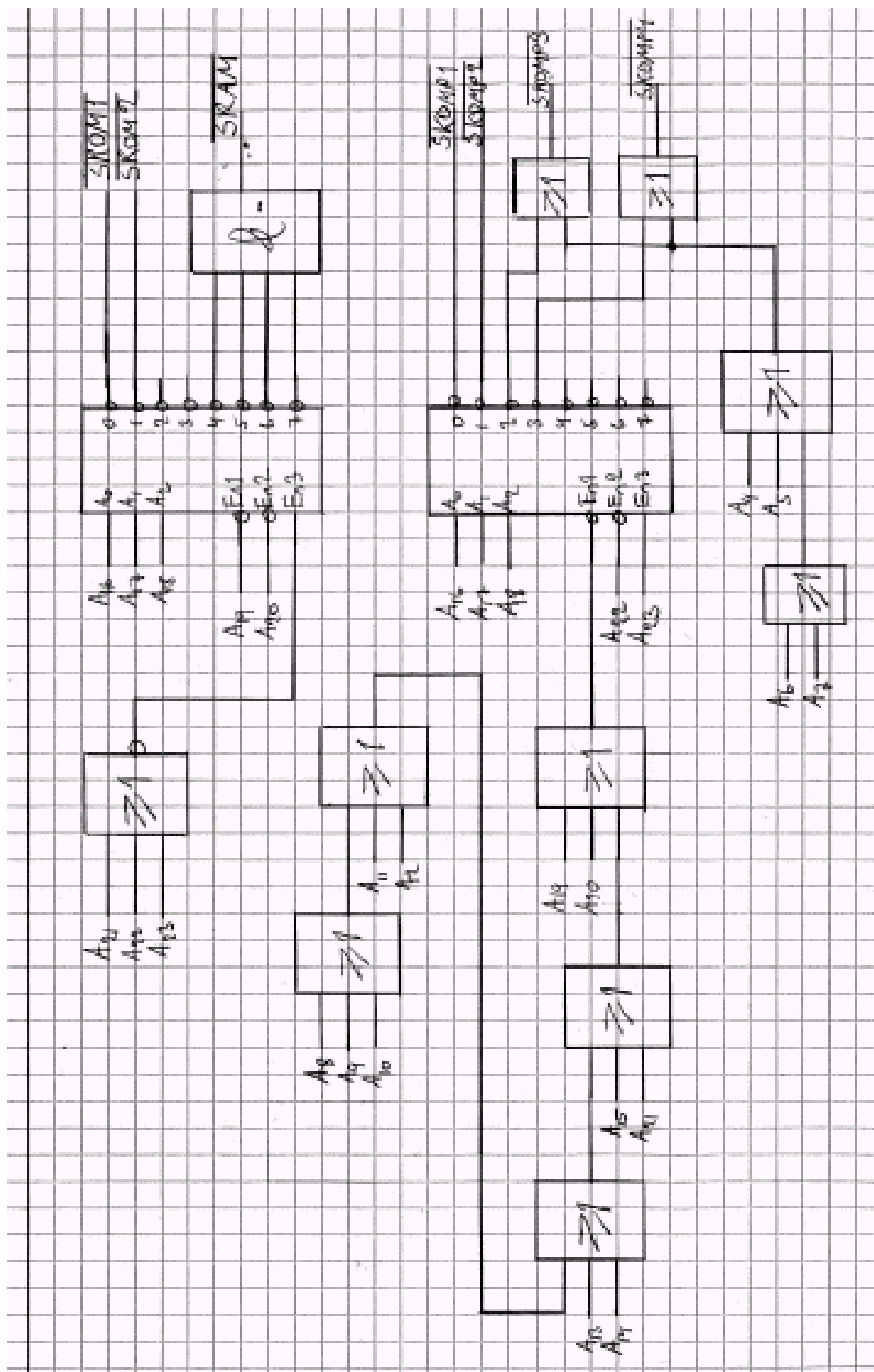


- 10. A) Konstruera en fullständig adressavkodare (fullständig adressavkodning) som realiserar adresskartan definierad i uppgiften. Implementera med hjälp av logiska grindar och avkodare (specificerad i Appendix)**

<u>Komponent</u>	<u>Adressområde</u>
ROM1	00 0000 - 00FFFF
ROM2	010000 - 01FFFF
RAM	04 0000 - 07 FFFF
KOMP_1	80 0000 - 80 00FF
KOMP_2	81 0000 - 81 00FF
KOMP_3	82 0000 - 82 000F
KOMP_4	83 0000 - 83 000F

[illegible]

[illegible]



B) Hur många bytes är definierat för varje komponent i adresskartan?

Komponent	Adressbitar som omfattas av komponent	Storlek
ROM1	A ₁₅ - A ₀	2 ¹⁶ =2 ⁶ kbyte = 64 kbyte
ROM2	A ₁₅ - A ₀	2 ¹⁶ =2 ⁶ kbyte = 64 kbyte
RAM	A ₁₇ - A ₀	2 ¹⁸ =2 ⁸ kbyte = 256 kbyte
KOMP_1	A ₇ - A ₀	2 ⁸ byte = 256 byte
KOMP_2	A ₇ - A ₀	2 ⁸ byte = 256 byte
KOMP_3	A ₃ - A ₀	2 ⁴ byte = 16 byte
KOMP_4	A ₃ - A ₀	2 ⁴ byte = 16 byte

11. Skriv en subrutin som konverterar ett 32-bitars binärttal till en textsträng med nollor och ettor.

In: D0 = Talet som ska omvandlas. A0 pekar på startadressen för strängen, dvs på det mest signifikanta bit-tecknet.

Ut: En teckensträng med ettor och nollor som avslutats med en binär nolla i den sista positionen (A0+32) för att indikera strängslut.

Exempel: Talet \$4323 Ska ge textsträngen "00000000000000000100001100100011",0.

* Subroutine: binaryToString

*

* Input parameters: D0.L number to be converted into a string of '1' and '0'

*

A0 points to the start address of the string to be written

* Output: A null terminated string will be written at start address of A0

binaryToString:

MOVEM.L D0-D2/A0,-(SP)

* D0-D2 and A0 will be used and copied to the stack

MOVE.B #32,D1

* Load counter for 32 bits

Loop:

ROL.L#1,D0

* Rotate left to get next bit

MOVE.B D0,D2

* Take a copy

ANDI.B #1,D2

* Only consider one bit (bit 0) at a time

ADDI.B #\$30,D2

* Add the ascii code for a '0' to this bit

MOVE.B D2,(A0)

* Write this converted character to the string

ADDQ.L #1,A0

* Increment the pointer to next character to be written

SUBQ.B #1,D1

* count bits

BNE Loop

* branch as long as D1 > 0

MOVE.B #0,(A0)

* zero terminates the string

MOVEM.L (SP)+,D0-D2/A0

* Retrieve register D0-D2 and A0

RTS