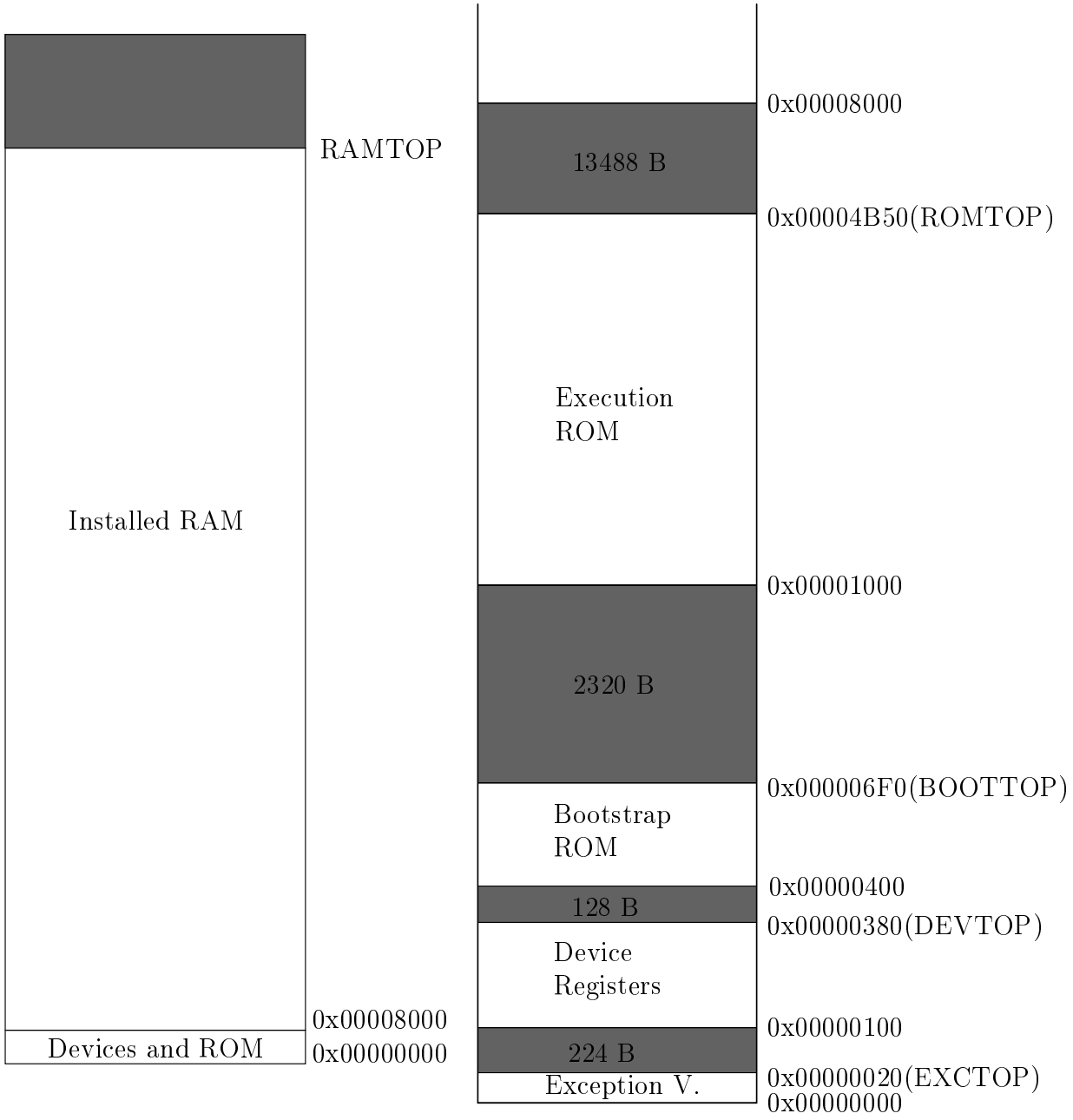
1 Memoria fisica

1.1 Proposta 1: ARM friendly

Il linker assume che su architettura arm l'indirizzo da cui inizia il caricamento degli eseguibili sia 0x00008000, quindi, considerata la dimensione in memoria delle ROM di umps e lo spazio occupato da exception vector e device registers e la disposizione più sensata sembra essere (come mostrato in figura):

- il **vettore delle eccezioni** posizionato all'indirizzo 0x00000000, come da specifiche del processore;
- i device sono 40, ognuno rappresentato da 4 registri, per un totale di 640 Bytes occupati dai **registri dei device**, per semplicità questa area di memoria inizia all'indirizzo 0x00000100;
- la **ROM di bootsrap** può essere posizionata all'indirizzo tondo subito successivo ai device poichè la dimensione di quell'area di memoria è fissa, quindi inizierà da 0x00000400;
- la dimensione dell'eseguibile di bootstrap che in umps si occupa del caricamento del sistema operativo da tape (quella più ingombrante) richiede 640 Bytes, si lasciano liberi altri 2320 Bytes per permettere l'inserimento di una ROM più complessa. Il **BIOS** verrà quindi caricato a partire dall'indirizzo 0x00001000, ed avrà a disposizione un totale di 28.672 Bytes (il BIOS di umps ne occupava 15184);
- la **RAM** disponibile in lettura/scrittura partirà quindi dall'indirizzo 0x00008000 come suggerisce lo script di ld per arm.

Nello schema vengono specificati come valori di ROMTOP e BOOTOP i valori che sarebbero assunti utilizzando le ROM di umps in modo da avere una prima idea dello spazio riservato.

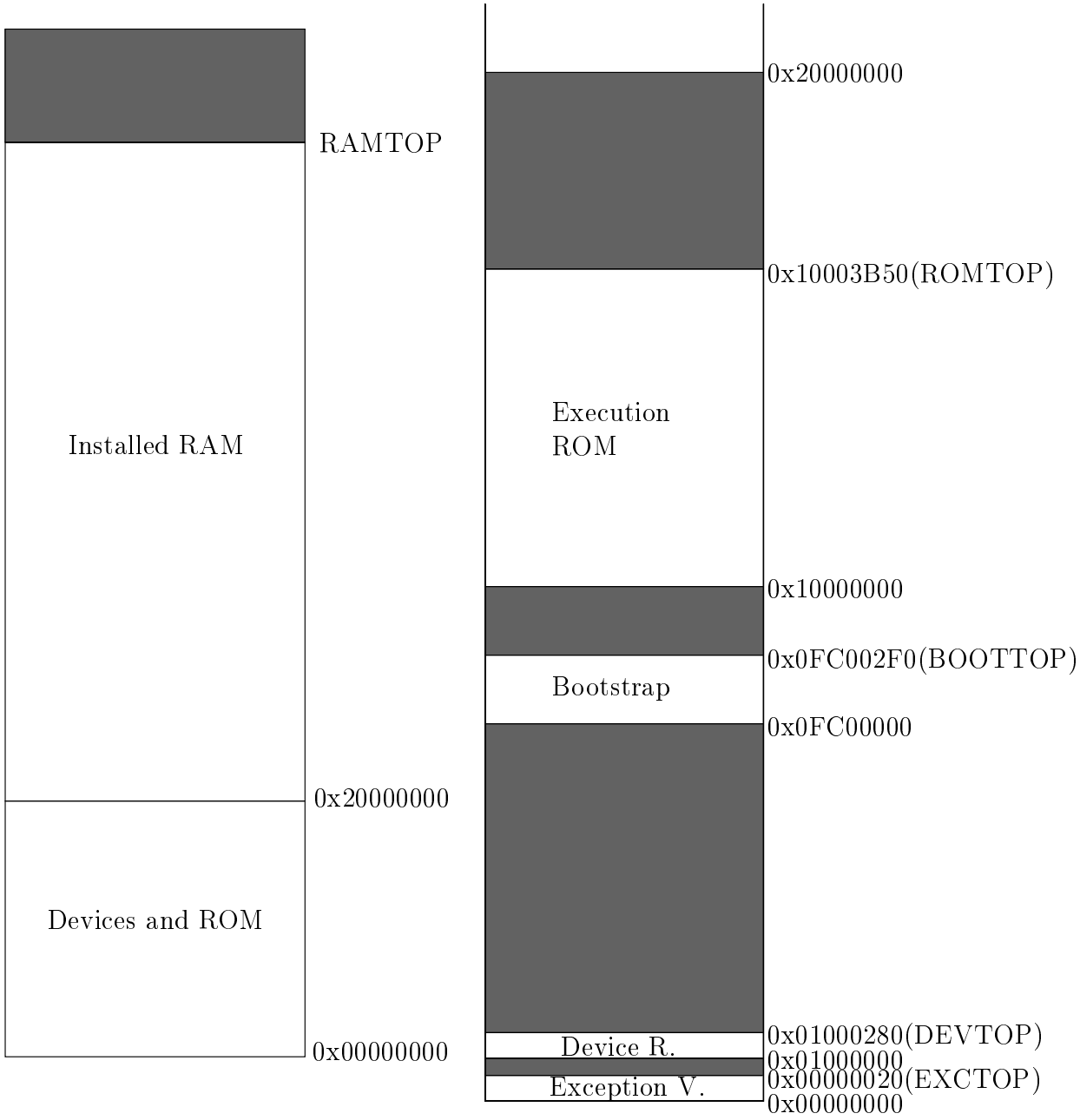

RAMTOP
Installed RAM
0x00008000
Devices and ROM
0x00000000
0x00008000
13488 B
0x00004B50(ROMTOP)
Execution
ROM
0x00001000
2320 B
0x000006F0(BOOTTOP)
Bootstrap
ROM
0x00000400
128 B
0x00000380(DEVTOP)
Device
Registers
0x00000100
224 B
0x00000020(EXCTOP)
Exception V.
0x00000000

1.2 Proposta 2: uMPS friendly

Per sfruttare le caratteristiche (e il codice) di umps, si può procedere modificand lo script di linking in modo da avere un'architettura più simile a quella proposta dall'emulatore: i primi 0.5 GB vengono riservati per ROM e device registers (ed exception vector) per fare indirizzamento e check di segmento più efficienti.

Ne deriva una struttura come segue (mostrata sotto in figura):

- il **vettore delle eccezioni** continua a risiedere all'inizio dello spazio degli indirizzi poichè il processore richiede che si trovi in questa locazione
- i **registri dei device** saranno collocati all'indirizzo 0x01000000
- la **ROM di bootstrap** verrà collocata all'indirizzo 0x0FC00000
- in ultimo la **ROM del BIOS** viene caricata a 0x10000000
- la **RAM** avrà quindi disponibile lo spazio degli indirizzi da 0x80000000 fino a *RAMTOP*


RAMTOP
Installed RAM
0x20000000
Devices and ROM
0x00000000
0x20000000
0x10003B50(ROMTOP)
Execution
ROM
0x10000000
0x0FC002F0(BOOTTOP)
Bootstrap
0x0FC00000
0x01000280(DEVTOP)
Device R.
0x01000000
0x00000020(EXCTOP)
Exception V.
0x00000000

1.3 Note

Entrambe le proposte necessitano la creazione di un file .x apposito, la prima è più aderente ai metodi standard impiegati dal linker arm, circa come avevamo pensato. Impiegando la seconda proposta però si ha il vantaggio di dover intervenire in maniera meno drastica su elf2umps per preparare gli eseguibili più ulteriori vantaggi quando si aggiunga la memoria virutale.

2 Memoria Virtuale

Lo schema di umps rimane appropriato, l'unica modifica degna di nota è nel frame riservato per la ROM (che veniva originariamente posizionato da 0x20000000 a 0x20000FFF): umps riserva tre porzioni di memoria, in questo frame, per vettori delle eccezioni e degli interrupt, segment table e stack.

Arm richiede che il vettore delle eccezioni sia all'inizio dello spazio degli indirizzi, quindi:

- nel caso in cui venga adottata la seconda proposta, si può mappare semplicemente l'intero frame sugli indirizzi da 0x00000000 a 0x00000FFF in modo da allocare anche il vettore degli interrupt consecutivamente a quello delle eccezioni;
- se invece si adotta la prima proposta si può mappare il frame all'inizio della ram, quindi da 0x00008000 a 0x00008FFF, sostituendo alla prima sezione il solo vettore degli interrupt.

2.1 MMU

Come già umps, si tende ad evitare di implementare una vera mmu.

La memoria viene divisa in 64 segmenti, uno per ogni ASID, ogni segmento specifica le proprie page table di massimo 1M (una per ogni macrosegmento: *ksegOS*, *kUseg1* e *kUseg2*), la traduzione degli indirizzi è effettuata dalla cpu in congiunzione al coprocessore di sistema:

- CP0 contiene informazioni sull'ASID attuale, in base a questo valore e al numero di segmento (che se si impega la prima proposta per la memoria fisica, rimane come in umps e quindi occupa i primi due bit dell'indirizzo virtuale) viene raggiunta la page table corretta.

- CP0 legge la parte alta dell'indirizzo virtuale in modo da fare il check sulla page table in base al virtual page number. In caso di successo restituisce l'indirizzo fisico associato.

- La CPU riceve il physical frame number e gli somma l'offset estratto dalla parte bassa dell'indirizzo virtuale e quindi accede alla memoria.

Siccome arm non forza una struttura specifica per il coprocessore di sistema (*CP15*), potrebbe essere una buona idea reimplementare la parte che si occupa della gestione della memoria del CP0 di umps in una versione analoga compatibile con gli standard consigliati da arm.